August 1981

Report. No. STAN-CS-81-869

Computer Science Comprehensive Examinations 1978/79 - 1980/81

edited by

Carolyn E. Tajnai

Department of Computer Science

Stanford University
Stanford, CA 94305

Computer Science Comprehensive Examinations 1978/79 - 1980/81

edited by

Carolyn E. **Tajnai**

Department of Computer Science

Stanford University
Stanford, CA 94305

Abstract

The Stanford Computer Science Comprehensive Examination was conceived Spring **Quarter 1971/72** and since then has been given winter and spring quarters each year. The *Comp* serves several purposes in the department. There are no course requirements in the Ph.D. and the Ph.D. Minor programs, and only one *(CS293, Computer Laboratory)* in the Master's program. Therefore, the *Comp* fulfills the breadth and depth requirements. The Ph.D. Minor and Master's student must pass at the Master's level to be eligible for the degree. For the Ph.D. student it serves as a "Rite of Passage;" the exam must be passed at the Ph.D. level by the end of six quarters of full-time study (excluding summers) for the student to continue in the program.

This report is a collection of comprehensive examinations from Winter Quarter **1978/79** through Spring Quarter 1980181.

Foreword

In November, 1978, Frank Liang published the first collection of Computer Science Department Comprehensive Examinations, *STAN-CS-75-677,* and the document proved to be a tremendous success. No attempt has been made to emulate Frank's style; this collection is strictly utilitarian.

The comprehensive examination serves several purposes in the department. There are no course requirements in the Ph.D. and the Ph.D. Minor programs, and only one *(CS293, Computer Laboratory) in the* Master's program. Therefore, the comprehensive fulfills the breadth and depth requirement. The Ph.D. Minor and Master's student must pass the exam at the Master's level to be eligible for the degree. For the Ph.D. student it serves as a "Rite of Passage;" the exam must be passed at the Ph.D. level by the end of six quarters of full-time study (excluding summers) for the student to continue in the program.

The written portion is a six-hour examination given winter and spring quarters. Until January, 1979, the programming portion was a 5-day take-home project given the week after the written portion, and the two were graded together. At the June 13, 1978, Faculty Meeting it was decided to separate them; the grading would be independent.

During 1979/80 the Comprehensive Examination Committee became aware of the difficulty of equitably grading a program written during a high stress, five day period. The following motion was passed at the June 10, 1980, faculty meeting.

> Professor (Michael) Genesereth, representing the Comprehensive Examination Committee, proposed that the following procedure be adopted for the Comprehensive Programming Problem.
>
> "Students in the M.S. and Ph.D. programs (and Ph.D. Minor students who have passed the written examination) in Computer Science must prepare a programming project of sufficient complexity and quality to demonstrate competence in computer programming.
>
> This project must be supervised and endorsed by a member of the Computer Science Department faculty and submitted to the Comprehensive Examination Committee for final approval. The project must be written at Stanford by the student, working independently.
>
> The project must exhibit the use of sophisticated algorithms and data structures and be well documented. Programs will be judged on the basis of correctness, efficiency, clarity, and style. The project may be the result of CS293 work, although it need not be. The project should represent at least 3 units of work."
>
> Professor Genesercth made a motion that the proposal be accepted; Professor (Forest) Baskett seconded the motion, and it was passed.

At the faculty meeting on June 9, 1981, new guidelines *Computer Science Department Requirements for the Comprehensive Programming Project* were adopted, for further clarification. See page *ix* of this report.

For those of you who are preparing to take the exam, lots of good luck.

Carolyn Taj nai
July 1981

Comprehensive Examination Reading List

(Revised August 21, 1981)

ALGORITHMS AND DATA STRUCTURES

Aho, A. V., Hopcroft, J. E., and Ullman, J. D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, 1974, Chapters 1, 2, 3, 4.1-4.4, 5.1-5.4. Chapter 10.1-10.5 covers some of the same material as Garey & Johnson (below).

Garey, M. R., and Johnson, D. S., Computers and *Intractability*, Freeman, San Francisco, 1978, Chapters 1-3.

Knuth, D. E., The Art *of* Computer Programming, Volume 1, Addison-Wesley, Reading, Massachusetts, 1968, Chapter 2 (except for Section 2.3.4).

ARTIFICIAL INTELLIGENCE

Barr, A.V. and E.A. Feigenbaum (eds.), Handbook of Artificial Intelligence, Volume 1., Kaufmann, Stanford, 1981.

Winston, P.H., *Artificial* Intelligence, Addison-Wesley, Reading, Massachusetts, 1977, Part I, Chapters 1–9.

HARDWARE SYSTEMS

General:

Mano, M., Computer System Architecture, Prentice-Hall, Englewood Cliffs, New Jersey, 1976, Chapters 1-5, 7, 8, 11.1, 11.2, 11.5, and 12; or you may substitute Gschwind, H. W. & McCluskey, E. J., Design of Digital Computers, Springer-Verlag, New York, 1975, Chapters 2, 3, 5, 6, 7, 8.2, 8.3 (except for 8.3.5.1), 8.4.

Memory Hierarchy:

Matick, R., Computer Storage Systems and Technology, Wiley Interscience, Chapter 9.

Strecker, W. D., Cache Memories *for PDP-11* Family Computers, 3rd Annual Computer Architecture Symposium.

Computer Systems:

C. A. C. M. Jan 1978: CRAY-1 , pages 63-72; *IBM* 370, pages 73-96.

Stack Computers:

Stone, H. *Introduction to* Computer Architecture, SRA, 75, Chapter 7.

I/O:

Kraft, G. D. and Toy, W. N., Mini/Microcomputer *Hardware* Design, Prentice-Hall, 1979, Chapters 3, 5, 6, 8, 9.

NUMERICAL ANALYSIS

Atkinson, K. E., *An* Introduction to *Numerical Analysis,* Wiley, New York, 1978, Chapters 1-3. Or you may substitute Conte, S. D., and De Boor, C., *Elementary* Numerical Analysis: *An* Algorithmic Approach, 2nd ed., McGraw-Hill, New York, New York, Chapters 1-2 and 4.1-4.8; or Conte and De Boor, 3rd ed., 1980, Chapters 1-3.

Forsythe, G. E., Malcolm, M. A., and Moler, C. B., Computer Methods for Mathematical Computations, Prentice-Hall, 1977, Chapters 2, 4.4, and 4.5.

Forsythe, G. E., and Moler, C. B., Computer Solution *of Linear Algebraic Systems,* Prentice-Hall, 1967.

SOFTWARE SYSTEMS

Aho, A. V., and **Ullman,** J. D., *Principles of Compiler Design,* Wiley, New York, New York, 1975.

Brinch Hansen, P., Operating System *Principles,* Prentice-Hall, 1973.

Dahl, O.-J., Dijkstra, E. W., and **Hoare,** C. A. R., *Structured Programming,* Academic Press, New York, New York, 1972.

Graham, R., *Principles of Systems Programming,* Addison-Wesley, Reading, **Massachusetts,** 1975.

Stone, H. **S.,** *Introduction to Computer Organization and* Data *Structures,* McGraw-Hi& New York, New York, 1972, Chapters 1-8. Contains basic knowledge about computer organization. Most students should just skim this.

Watson, R., *Timesharing System* Design *Concepts,* McGraw-Hi& 1970, section 2.4, or Denning, P., "Virtual Memory," *Computing Surveys,* September, 1970.

THEORY OF COMPUTATION

Hopcroft, J., and Ullman, J., *Introduction to Automata* Theory, Languages, *and Computation,* Addison-Wesley, 1979, Chapters 1-3, 4.1-4.6, 5-7, 8.1-8.5.

Manna, **Z.,** *Introduction to Mathematical Theory of Computation,* McGraw-Hi& 1973, Chapters 1, 2 and 3. Alternative introductions to logic appear in Mendelson, E., *Introduction to Mathematical Logic,* Van Nostrand, Chapters 1-2, or Enderton, H., *A Mathematical Introduction* to *Logic,* Academic Press, 1973, Chapters 1-2.

McCarthy, J., and **Talcott, C.,** *LISP: Programming* and *Proving,* (available from Stanford Bookstore) **1980,** Chapters 1-3.

RECOMMENDED COURSES

The Comprehensive Exam is meant generally to cover the material from the following courses: CS 111 (assembly language); 311 (hardware) 137A (numerical analysis); 107, 142, 143, and 246A (systems); **144A,** B (data structures); 156 (theory of computation); and 223 (artificial intelligence). Since the precise content of these courses varies somewhat, the actual scope of the Exam will be determined by the references above. Please note that the reading list includes some material involving structured programming as well as the history and culture of Computer Science even though it does not correspond to any **particular** course.

The Exam will also assume a certain mathematical sophistication and a knowledge of programming. The mathematical sophistication required may include knowledge of techniques such as induction, recursion, "divide and conquer" (e.g., techniques in sorting algorithms, case arguments, etc.), and will be at the level of an upper division undergraduate in the mathematical sciences. Proofs of correctness for simple programs may be required. The programming knowledge required will be an ALGOL-like language (e.g., Pascal), a knowledge of LISP, and possibly some assembly language. The exam will be "open-book-and-notes." This means you are allowed to use any materials you bring with you, plus copies of the above materials which will be made available. Non-smoking and smoking examination rooms will be scheduled. Copies of previous exams are available from the department.

PROGRAMMING PROBLEM

Students in the M.S. and Ph.D. programs (and Ph.D. Minor students who have passed the written examination) in Computer Science must prepare a programming project of sufficient complexity and quality to demonstrate competence in computer programming.

This project must be supervised and endorsed by a member of the Computer Science Department faculty and submitted to the Comprehensive Examination Committee for final approval. The project must be written at Stanford by the student, working independently.

The project must exhibit the use of sophisticated algorithms and data structures and be well documented. Programs will be judged on the basis of correctness, efficiency, clarity, and style. The project may be the result of CS 293 work, although it need not be. The project should represent at least 3 units of work.

The text

> Kernighan, B. W. and Plauger, P. J., ***The Elements of*** Programming ***Style,*** McGraw-Hi& New York, New York, 1974.

discusses some matters of style.

Good luck.

Computer Science Department

Requirements for the Comprehensive Programming Project

This memo specifies the requirements for the Comprehensive Programming Project. It is intended as a guide to students doing the Project, to Faculty members sunervisine them, and to future Comprehensive Committees.

The policy on the Project was set by the Faculty in a resolution reproduced below. This memo also details the present Comprehensive Committee's interpretation of the resolution, and the manner in which it will be implemented.

1. Faculty Resolution

   As stated in the Charge to the Comprehensive Committee (<CSD.FILES>COMPCHARGE.DOC), the following resolution was passed at the June 10, 1980 Faculty meeting:

   "Students in the M.S. and Ph.D. programs (and Ph.D Minor students, who have passed the written examination) in Computer Science must prepare a programming project of sufficient complexity and quality to demonstrate competence in computer programming.

   "This project must be supervised and endorsed by a member of the Computer Science Department faculty and submitted to the Comprehensive Examination Committee for final approval. The project must be written at Stanford by the student, working independently.

   "The project must exhibit the use of sophisticated algorithms and data structures and be well documcntcd. Programs will be judged on the basis of correctness, efficiency, clarity, and style. The project may be the result of CS293 work, although it need not be. The project should represent at least 3 units of work."

2. Complexity

   The project must involve both design of algorithms and data structures, and actual programming. It should bc such that the design aspect is significant. A program that is very long, but consists only of a large number of trivial algorithms and data structures, is not adequate. Also, implementation of a program from Someone else's design is not adequate.

   As a guidclinc, a program that correctly and completely solves one of the Comprehensive Programming Problems set between 1972 and 1980 would be considered sufficiently complex. (Some of these were published in Tech. Report STAN-CS-78-677; the others may be obtained from Carolyn Tajnai). However, these specific problems arc not acceptable as projects because full solutions to them have been published.

3. Quality

Projects will be judged on quality of both code and documentation. The judgement of code will be based on correctness, clarity, style and efficiency. A program should be easily readable by an experienced programmer conversant with the language used.

The importance of good documentation cannot be emphasized too strongly. Both internal and external documentation are essential. Between them, they should clearly and concisely state at least the following:

a. The purpose of the project: the problem it solves or the service it provides.

b. The architecture of the solution: program structure, major data structures, and the relationships beween them.

c. Design decisions taken, alternatives considered and the rationale behind the choices made. Reasons for choosing particular algorithms and data structures should be given. Clarity/efficiency, space/time and other tradeoffs should be documented and justified.

d. The implementation: how data structures are implemented and details of algorithms used.

e. Details of test runs performed and the results produced. Testing should be sufficient to demonstrate that the project achieves its stated purpose.

f. Citations and acknowledgements of all literary material used and all advice received from others (see section 4).

Verbosity should be avoided; 8 to 10 pages of external documentation should normally suffice.

It is up to the student in the documentation not only to make clear what was done and how it was done, but also to give some evidence that the way it was done is superior to alternative possibilities. The documentation should be written for a person having a good general knowledge of Computer Science and programming, but no specific knowledge of the particular project or project area.

Copies of projects of adequate complexity and quality may be obtained from Carolyn Tajnai. Candidates are strongly advised to examine these.

4. Use of Projects Written for Other Purposes

A project written for some other purpose may be submitted as a Comprehensive Project, provided it is written while the author is a student at Stanford. In particular the following are acceptable:

a. A course project;

b. A 293 project:

c. A project done as part of a Research Assistantship.

However, the evaluation of the project as a Comprehensive Project is entirely independent of any other evaluation. For example, it is conceivable that a course project earning an "A" would be insufficient to satisfy the requirements of the Comprehensive.

A program that is part of a larger program or system is acceptable as a Comprehensive Project, provided:

a. The portion submitted is complex enough by itself to satisfy the requirements given above.

b. The portion submitted can be run and tested. The fact that the state of the rest of the system prevented running and testing is not acceptable as an excuse.

5. Obtaining Advice

A student may consult the literature or seek advice on aspects of his/her project if necessary. However, all assistance and sources of information must be acknowledged in the documentation.

Receiving unacknowledged information or advice constitutes an Honor Code violation. Receiving and acknowledging an excessive amount of assistance is not a violation, but may lead to the project being rejected as inadequate. A student in doubt as to how much assistance is reasonable, should consult the Faculty member supervising the project or a member of the Comprehensive Committee.

If the faculty advisor has any questions related to the suitability of the project he/she should consult the Comprehensive Committee.

6. Administration

The following procedure should be followed by a student wishing to take the Programming Project:

a. Arrange to do the project under the direction of a Faculty member. Both the student and Faculty member should ensure that the project satisfies the requirements stated above.

b. On completion of the project to the Faculty member's satisfaction, obtain a Project Submission Form from Carolyn Tajnai, fill it in and have the Faculty member sign it. The form states that the project is the student's own work and that, in the opinion of the Faculty member, it is adequate.

c. Hand the signed form and the project to Carolyn Tajnai, who will pass it on to the domprchensivc Committee and eventually communicate the grade to the student. A graded project may be examined, but not kept, by the student; it will be kept on file for three years by the Department.

7. Grading

The Committee will grade projects as expeditiously as possible. In particular, it guarantees:

a. To examine a project and give an "immediate response" within ten working days. This response will be either a grade. if the grade is not in doubt, or a statement that the Committee requires more time to consider the project.

b. To grade any project handed in during the first two weeks of any quarter (including Summer) in time for graduation at the end of that quarter.

These are minimum performance guarantees; the Committee will always endeavour to better them.

As with the written Comprehensive Exam, the Programming Project can be passed at the MS or PHD level. If a project is not considered worth a pass at the level required by the student, it will be returned with comments, and the student will normally be given the opportunity to improve and resubmit it. In the case of a project that is wholly inadequate or is submitted more than three times in all, however, the Committee may require the student to undertake a completely new project.

This document was approved by the Faculty of the Department of Computer Science at the June 9, 1981, meeting.

Table of Contents

Winter Quarter 1978/79
Written Examination 1
Solutions 16
Programming Project 34

Spring Quarter 1978179
Written Examination 37
Solutions 49
Programming Project 63

Winter Quarter 1979/80
Written Examination 71
Solutions 88
Programming Project 106

Spring Quarter 1979180
Written Examination 113
Solutions 128
Programming Project 149

Winter Quarter 1980181
Written Examination and Solutions 155

Spring Quarter 1980/81
Written Examination and Solutions 183

Instructions and **Honor Code** 215

Theory *of* Computation

1. Turing (5 points)

(a) Why did A. M. Turing invent the "Turing machine"?

(b) Did he spend more *years* of his life working with abstract "Turing machines" or with real computer61 (Give some background information to support your answer.)

2. Resolution and Unification (7 points)

Prove by resolution that the following set of clauses is unsatisfiable.

$P(g(z, w), z, w)$

$\neg P(x, y, u) \vee P(y, z, v) \vee \neg P(x, v, w) \vee P(u, z, w)$

$\neg P(k(x), x, k(x))$

3. Sorted Languages (21 points)

Consider strings *over* an alphabet $\{a_1, \ldots, a_n\}$ whose letters are linearly ordered: $a_1 < \ldots < a_m$. If $\alpha = x_1 x_2 \ldots x_n$ is a string, let sorted(a) be the string $x_{p_1} x_{p_2} \ldots x_{p_n}$, where $p_1 p_2 \ldots p_n$ is a permutation of $\{1, 2, \ldots, n\}$ and $x_{p_1} \leq x_{p_2} \leq \ldots < x_{p_n}$.

If L *is* a language over $\{a_1, \ldots, a_m\}$, define new languages a6 follows:

$$\text{sortedsubset} = \{a \in L \mid a = \text{sorted(a)}\};$$
$$\text{sorted(L)} = \{\text{sorted(a)} \mid \alpha \text{ EL}\};$$
$$\text{unsorted(L)} = \{a \mid \text{sorted(a)} = \text{sorted(P) } \textit{for some } \beta \in L\}.$$

Prove *or* disprove the following statements:

(a) If L is context-free then sortedsubset(L) is context-free,

(b) If L is context-free then sorted(L) is context-free.

(c) If L is context-free then unsorted(L) *is* context-free.

4. Context sensitive grammar 6 (12 pints)

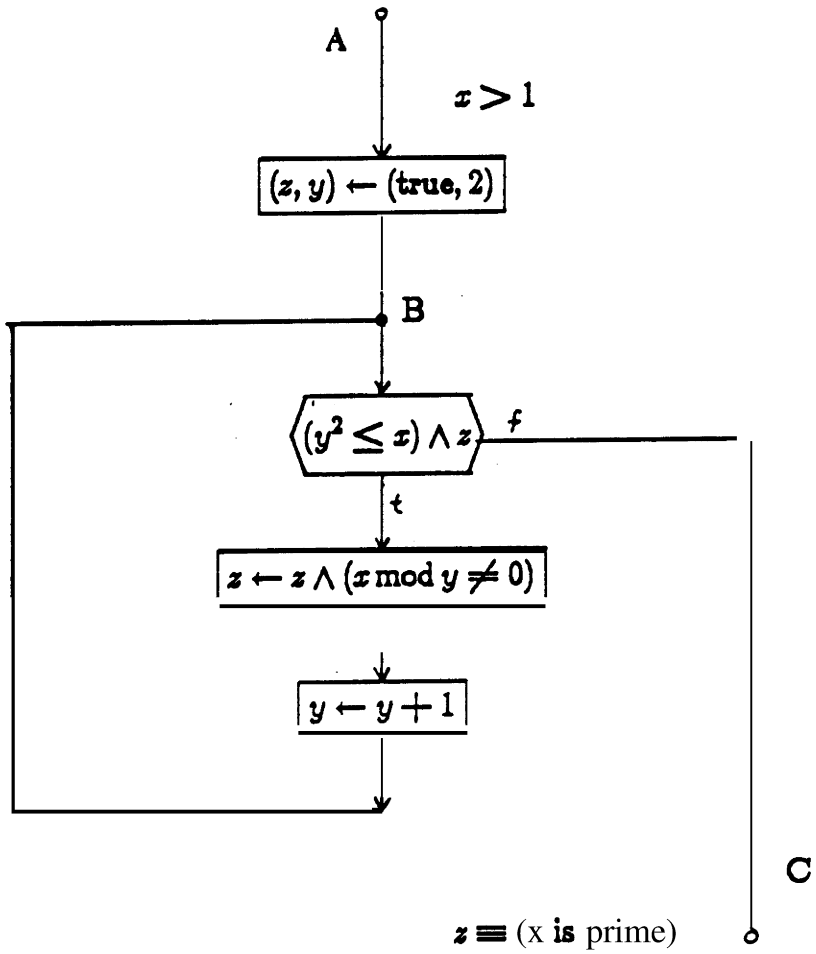
Determine the language generated by the following grammar. (Upper case letters are nonterminals, lower case letters *are* terminals, and S is the start symbol.)

$$S \to PABQ$$
$$PA \to PCT$$
$$TA \to BCT$$
$$TB \to BT$$
$$CB \to BC$$
$$TQ \to UQ$$
$$CU \to UB$$
$$BU \to VA$$
$$BV \to VA$$
$$PV \to PA$$
$$PA \to aX$$
$$XA \to aX$$
$$XB \to bX$$
$$XQ \to qq$$

Hint: Consider the strings derivable *from* PA^nB^mQ.

5. **Program Verification** (15 **points**)

Invent a **suitable** inductive assertion at point B and prove the following program partially **correct** with respect to the given input and output predicates. Generate and **prove all** verification conditions.



Artificial Intelligence

1. Theorem proving (5 points)

It ha6 been suggested that work on theorem proving ha6 been shelved temporarily. Supposing thie is correct, what would be the reason for this trend? State your answer briefly.

2. **Production systems** (5 points)

Comment briefly on the differences between production system architecture6 when used for (a) psychological model6 of cognitive skill6 (such a6 PSG) and (b) expert systems (such as MYCIN or AM).

3. Performance (5 points)

Pick ONE of the pairs of program6 listed below and contrast the approaches used in the two programs of that pair. In light of the superior performance of the "less intelligent" program, defend the continued use of AI in such problem areas.

(a) **HEARSAY** - DRAGON

(b) CHESS 4.6 [or *4.5* or 4.7] - *CAPS*

(c) INTERNIST [Pople's early version] - MYCIN

4. Choice *of* Task Domain (9 points)

Order the tasks below by the time it will take to produce commercial robots to do them. State the general principle6 you use to make the ordering and explain any exceptions.

Planning a meal

Cooking a meal

Serving a meal

Teaching arithmetic

Teaching soccer

Teaching (about) Shakespeare

5. **Concepts** (9 points)

Briefly define each *of* the *following* **AI** concept6 and methods, and give a one or two sentence description *of* **the** conditiona under which it **is** relevant:

actors

alpha-beta technique

British Museum algorithm

goal-directed search

LISP

Simon'6 ant

6. Games (27 points)

Consider the following problem:

You and an opponent **are** facing *11* stacks *of* pennies, *of* heights *11,10,9,...,1.* *You* will *alternate moves, removing* pennies, *and each time* someone takes *the* final *penny* in a *stack* his OPPONENT **will receive** one point. During his turn, each *player must* remove *three pennies (three from one pile, two from one pile* and *one from another, or one* **each from three separate piles).** What *should be* **your** *first move?* (Assume *that your opponent will play perfect&, that* **you** are *trying to* maximize *the number of points* **you will** receive, and that *your program* can have as much *time* and space as it calls for.)

(a) [10 points] Sketch the body of a recursive program to solve this problem. You may omit the details, and **use** math notation and concepts liberally.

(b) [7 points] Now fill in the details of the **above** sketch, such as the base steps of the recursion and the initialization **of any** necessary **variables** and data structures.

(c) [2 points] What language might be appropriate to implement this program in (very briefly mention why)?

(d) [4 points] Assume that, rather than being infallible, your opponents are many and varied in their skill. How might "intelligence" be inserted into the program so that it might attain very high scores?

(e) [4 points] How might *a* software analogue of "caching" be used to improve the program's efficiency? (If you prefer, you may answer this question using the software analogue of any other hardware concept.)

Systems

1. Compiler runtime organization (15 points)

Suppose you are writing an Algol compiler for some machine whose instruction set you know. Sketch how you would implement run-time display management on this machine. Where would you store the stack pointer, display pointers, and other necessary information? What would a stack frame look like?

2. One-pass, multi-pass compilers (5 points)

(a) What are the advantage6 of multi-pass compilers over one-pass compilers?

(b) Describe a way to handle code generation for forward jumps in a one-pass compiler for the generation of code in-core and for the generation of relocatable code on a file.

3. Exponentiation (12 points)

(a) [10 points] Pascal is sometimes criticized for lacking an exponentiation operator. Suppose you intended to add this operator (denoted **) to the language. Define precisely the meaning of *a**b*, where *a* and *b* are integer or real expressions. When will *a**b* be illegal (undefined)? What will be its type and value when it is legal? There is no single answer-try to make reasonable choices.

(b) [2 points] Based on your answer above, does the usefulness *of* the exponentiation operator justify the complexity it entails?

4. Parameter passing (8 points)

Suppose you are given a compiler for an Algol-like language. The language does not allow to specify in which way parameter6 are to be passed but you know that the compiler uses the same mechanism for all parameter types. Write one or more program fragment6 to determine whether parameter passing is

— call by value

— call by value result

— call by reference

— call by name.

Indicate how the answer can be derived from the result *of* your program.

5. **Banker's Algorithm** (5 points)

What is the purpose *of* the **Banker's** Algorithm? What information does it require?

6. **Synchronization (5 points)**

A critical region of a concurrent program may be executed only when no other critical region is being executed. Let us analogously define a serious region, which allows possibly one other **serious** region to be executed simultaneously.

Write an **Algol-like** program implementing this "almost" mutual exclusion for process i of n processes, each with one serious region. Show formally that your solution is correct in that (a) no more than two processes can execute their serious regions simultaneously, and (b) if fewer than two processes are in their serious regions, and other processes are waiting to enter their serious regions, then one will eventually be allowed to enter.

You may use shared variables, semaphores, critical regions, and conditional critical regions in your solution.

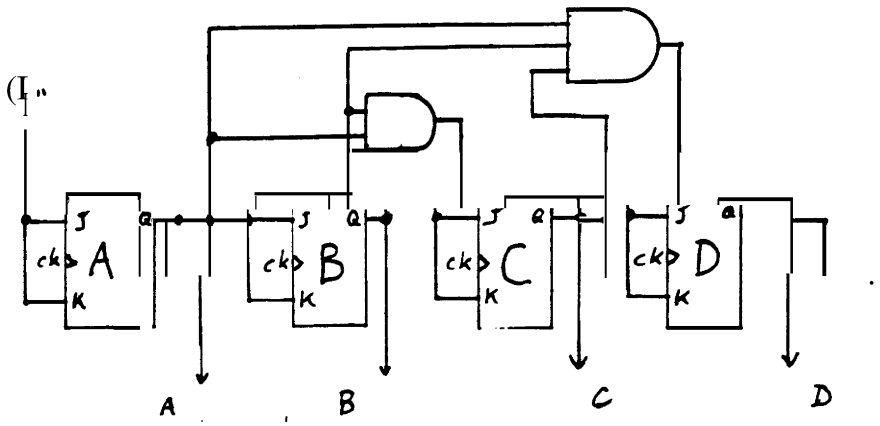
7. The **Class** Concept (10 points)

Write a Simula class implementing a bounded stack of integers. It should be possible to specify the stack limit when individual stacks are created. Provide the operations *push*, *pop*, and ***test-for-empty-stack***, assuming that stack overflow and underflow never occur. Show how to create and use instances of this class in a program.

Hardware

1. Logic design (33 points)

(a) [5 points] Write the state table of the following circuit:



(b) [8 points] Design a counter with the same state table, minimizing the number of gates.

(c) [15 points] Design a synchronous counter with the same state table, using D flip-flops.

(d) [2 points] Give 2 advantages of synchronous counters over asynchronous ones.

(e) [3 points] Give three reasons why one combinational circuit may be preferable to another requiring fewer gates.

2. Logic Technology (7 points)

Briefly describe each of the following logics:

RTL

MOS

ECL

TTL

Schottky

Josephson Junction

I I L

3. **Architecture** (15 points)

(a) [1 point] What is **a stack** machine?

(b) [1 point] What **is** *a* **register** machine?

(c) [3 points] What are each's **advantages** *over* the other?

(d) [10 points] Sketch how you would organize the CPU of **a** stack machine. Draw a block diagram showing the major components and their interconnections. It should be detailed enough to reveal how the stack is implemented in terms of other components.

4. **Celebrities** (5 **points)**

Name an accomplishment of each of the following persons:

M. Wilkes

T. Kilbum

S. Cray

G. Amdahl

G. Bell

1. Tree traversal (24 points)

(a) [15 points] The non-recursive procedure shown below performs an ***inorder*** traversal of a binary tree without using a stack. However, certain key parts of the procedure have been left out. You are to fill in the blanks by figuring out how the algorithm works.

The tree is represented in the usual manner, with each node having pointers to its left and right sons. The algorithm work8 by modifying certain pointers in the tree and later restoring them. **When** the traversal is completed, the *tree* has been returned to its original state.

```
t ← root;
while t ≠ Λ do begin
      if left(t) = Λ then begin
            visit(t); t ← [          ]
      end
      else begin
            p ← left(t);
            comment p is a temporary variable used only in this block;
            while right(p) ≠ Λ and right(p) ≠ [          ]
                  do p ← right(p);
            if right(p) = A then-begin
                  comment modify tree link to "remember our place";
                  right(p) ← t; t ← left(t)
            end
            else begin
                  visit(t);
                  comment fix up tree link;
                  right(p) ← [          ]; t ← [          ]
            end
      end
end;
```

(b) [9 points] Now modify the above procedure so that it performs a *preorder* traversal of the tree. Try to make as few changes as possible.

2. Data structures (20 points)

For each of the situations described below, you are to design a data structure to represent the set of values so that the indicated operations can be performed quickly. You should briefly describe how the operations are to be performed using your data structure, and estimate the running time.

(a) [8 points] We are given a set of numbers, and we want to perform the following operations:

(1) Add a number to the set, where this number is known to be larger than all of the numbers currently in the set.

(2) Delete the smallest number from the set.

(3) Delete the median number from the set. (In other words, if there are n numbers currently in the set, delete the $\lceil n/2 \rceil$ th smallest number.)

For parts (b), (c), and (d), consider the following situation. We have a supply of jars with specified capacities. We want to perform the following operations:

(1) Add a new jar of a specified capacity to our supply.

(2) Given a volume of liquid, find the *smallest* jar that can hold this volume. This jar is then deleted from our supply.

Answer the question for each of the following cases:

(b) [4 points] Jar capacities and liquid volumes are real numbers $\in [1, 50]$.

(c) [4 points] Jar capacities are real numbers; liquid volumes are integers $\in [1, 50]$.

(d) [4 points] Jar capacities are integers $\in [1, 50]$; liquid volumes are real numbers.

3. **Register** allocation (18 pints)

Consider a hypothetical computer with the following instructions:

ri ← m	load register **i** from memory location **m**
ri ← ri **o m**	register i gets the result of **ri** *o* **m**
ri ← *ri* **o rj**	register i gets the result of **ri o rj**

where **o** is a binary operation.

(a) [12 points] Suppose we are given a parenthesized expression involving only distinct variables (memory locations) and the **operator o,** for example

$$((a \circ b) \circ c) \circ (d \circ ((e \circ f) \circ g)).$$

We want to determine the *minimum* number *of* registers that are needed to compute the value of this expression.

Since the variables are distinct, you need not worry about common **sub-**expressions. You may use commutativity of the operator **o,** but do not assume associativity. Also, assume that the computer has an infinite number of registers, whose contents are initially undefined.

Give a formula for the minimum number of registers required by a given expression, and explain how the computation should be arranged to achieve this minimum number. You may introduce any additional notions that are appropriate.

(b) [4 points] Now assume that the machine has only k registers. What is the length (number of operations) of the shortest expression that cannot be computed by this machine?

1. Stable Algorithms, **well-conditioned** Problems (21 points)

In numerical computation, it is important to distinguish between an **ill-**conditioned problem and an unstable algorithm. In general, a problem is *ill-*conditioned if a small change in the data defining the problem results in a large change in the solution. An algorithm *is numerically* unstable if it introduces large errors in the computed solutions to problems which are not **ill-conditioned.** Note that conditioning is a property of the problem itself and that stability is a property of the particular method used to solve the problem. Here is a list of common numerical problems and possible methods for solving them. For each case, choose one of the following which comes closest to describing the situation. Briefly explain your conclusions by providing examples, pointing to error analyses, etc.

"Good-good": Well-conditioned problem and stable algorithm.
" Good-bad" : Well-conditioned problem and unstable algorithm.
'Bad-good" : Ill-conditioned problem and stable algorithm.
'Bad-bad": Ill-conditioned problem and unstable algorithm.

(a) Integration of a smooth function $f(x)$ *over* $[0, 1]$ using Simpson's rule with equally spaced points.

(b) Differentiation of the same function using finite differences with equally spaced points.

(c) Computation of the **roots of** f(x) $= x^5 - 5x^4 + 9x^3 - 7x^2 + 2x$ using Newton's method. Note that $f(x) = x(x-1)^3(x-2)$.

(d) Inversion of the matrix

$$A = \begin{pmatrix} .0001 & 1 \\ 1 & 2 \end{pmatrix}$$

using Gaussian elimination with no pivoting.

(e) Inversion of the same matrix using Gaussian elimination with partial pivoting.

(f) Inversion of the positive definite matrix

$$A = \begin{pmatrix} 1/19 & 1/18 \\ 1/18 & 1/17 \end{pmatrix}$$

using Gaussian elimination with no pivoting.

(g) Inversion of the same positive definite matrix using Gaussian elimination with complete pivoting.

2. One-sided **finite-difference** approximation (15 points)

(a) Find coefficients $\alpha, \beta,$ and 7 so that

$$y' = \alpha f(x) + \beta f(x + h) + \gamma f(x + 2h)$$

is a good approximation to the first derivative f(x). Note that this is a 'one-sided" approximation because no values of f to the left of x are used. Make whatever smoothness assumptions you think appropriate.

(b) Obtain an error bound of the form

$$|y' - f'(x)| \leq ch^k.$$

What are c and k?

(c) How small must h be in order that this formula can be used to compute $\cos(x)$ to four places of accuracy from a table of sin(x)?

3. **Decomposition** *of* a Matrix *(8* points)

Exercise 9.8 in Forsythe and **Moler,** *Computer Solution of* Linear Algebraic *Systems,* asks for a proof of the following "theorem".

Any symmetric, nonsingular matrix A can be expressed as a product

$$A = LDL^T$$

where L is a lower triangular matrix with positive diagonal elements, L^T is its transpose, and D is a diagonal matrix with ± 1 on the diagonal.

(a) Show by means of a simple counterexample that this "theorem" is false.

(b) What additional hypothesis on A would make the statement valid? (A hypothesis less strict than positive definiteness is possible.)

4. Representation of floating point numbers (8 points)

A new minicomputer, the Avon 9000, has an unorthodox arithmetic unit. When the following problem is executed, the operating system signals a division by zero. What base might be used for the representation of floating point numbers in the Avon 9000 firmware?

```
begin
    H: = 1.0/2.0
    x: = 2.0/3.0 - H
    Y: = 3.0/5.0 - H
    E: = (X + X + X) - H
    F: = (Y + Y + Y + Y + Y) - H
    Q: = F/E
    print Q
end
```

5. p-norm (8 points)

The **p-norm** of a vector x is defined for $1 \leq p < \infty$ by

$$\|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p}.$$

(a) What is

$$\lim_{p \to \infty} \|x\|_p \quad ?$$

(No proof required.)

(b) What is the purpose of the restriction $1 \leq p$?

(c) What difficulty with **floating-point** arithmetic might be encountered in a sub routine or procedure that computes the **p-norm** of a **vector** by directly implementing the definition?

Theory of Computation

I. (a) In order to prove that things were uncomputable by algorithms, it was desirable to have a simple device that could (in principle) compute all computable things.

(b) He designed ancestors of electronic computers during World War II, as part of his important code-breaking work, then he was chief architect of the Manchester-Ferranti machines in the late 40s and early 50s. Thus, by far the greatest part of his involvement was with concrete machines.

2. Set x t $g(Z, W)$, y $\leftarrow Z$, u $\leftarrow W$ *in* the second clause; resolve it with the first to get $P(Z, z, v) \vee \neg P(g(Z, W), v, w) \vee P(W, z, w)$. Now set v t Z, $w \leftarrow W$ and get $P(Z, z, Z) \vee P(W, z, W)$. Finally set $z \leftarrow x$, $Z \leftarrow$ k(x), $W \leftarrow k(x)$ and get $\emptyset$.

3. (a) True, since sortedsubset(L) = $L \cap a_1^* \ldots a_m^*$, and the intersection *of* context-free with regular is context-free.

(b) False; since $L = (abc)^*$ is regular, hence context-free, but sorted(L) = $\{ a^n b^n c^n \mid n \geq 1 \}$ is not context-free.

(c) False, by (a) and (b), since sorted(L) = sortedsubset(unsorted(L)).

4. Let $m, n \geq 1$. The only derivations from PA^nB^mQ that don't lead to dead ends essentially have the form $PA^nB^mQ \rightarrow PCTA^{n-1}B^mQ \rightarrow^* PC(BC)^{n-1}TB^mQ \rightarrow^* PC(BC)^{n-1}B^mTQ \rightarrow PC(BC)^{n-1}B^mUQ \rightarrow^* PB^{n-1+m}C^nUQ \rightarrow^* PB^{n-1+m}UB^nQ \rightarrow PB^{n-2+m}VAB^nQ \rightarrow^* PVA^{n-1+m}B^nQ \rightarrow PA^{n+m}B^nQ$, or the form $PA^nB^mQ \rightarrow aXA^{n-1}B^mQ \rightarrow^* a^nXB^mQ \rightarrow^* a^nb^mXQ \rightarrow a^nb^mqq$.

Thus, the terminal strings are all derived as follows, for some $k \geq 1$: S $\rightarrow PABQ \rightarrow^* PA^2BQ \rightarrow^* PA^3B^2Q \rightarrow^* PA^5B^3Q \rightarrow^* \ldots \rightarrow^* PA^{F_k+1}B^{F_k}Q \rightarrow^* a^{F_k+1}b^{F_k}qq$, where F_k denotes the kth Fibonacci number.

5. At point B we have "$x > 1$ and $(y-1)^2 \leq x$ and $z = \forall w(2 \leq w \leq (y-1) \Rightarrow x \bmod w \neq 0)$". Proof: The first time we get to B this is clearly true. Going backwards around the loop, before y $\leftarrow$ y + I we may assert "$x > 1$ and $y^2 \leq x$ and $z = \forall w(2 \leq w \leq$ y $\Rightarrow$ xmodw $\neq 0)$"; before $z \leftarrow z \wedge$ (xmody $\neq 0$) we have "$x > 1$ and $y^2 \leq x$ and $z = \forall w(2 \leq w \leq (y-1) \Rightarrow x \bmod w \neq 0)$". All verification conditions are trivial except we must show that "$x >$ I and $(y-1)^2 \leq x$ and $z = \forall w(2 \leq w \leq (y-1) \Rightarrow$ x mod $w \neq 0)$" and not "$(y^2 \leq$ x) $\wedge z$" implies "$z \equiv (x$ is prime)".

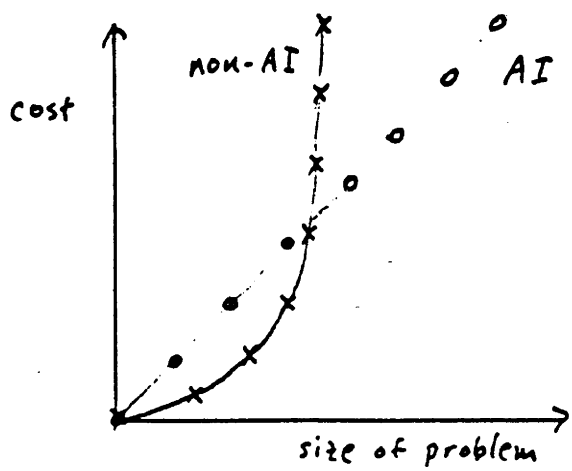
If not, we have one of two cases: (a) z false and x is prime. Then there is a w such that $2 \leq w \leq y-1 \leq \sqrt{x} < x$ and $x \bmod w = 0$, so w is a proper divisor of x; contradiction. (b) z true and x is not prime. Then, since x $>$ I, we have x = uv for some proper divisors u and v, where $2 \leq$ u $\leq$ v $\leq$ x; in particular, xmod u = 0. Therefore $2 \leq$ u $\leq$ y $-$ 1 is false, i.e., u $\geq$ y, and x = uv $\geq u^2 \geq y^2$; contradiction. . .

Artificial Intelligence

1. (1) The decade-old flurry of excitement over Robinson resolution subsided when few effective strategies were found for constraining the wmbinatorially explosive starch it entails. (2) Axiomatization of most problems is quite long and difficult, hence AI researchers are simply not able to bring predicate calculus theorem provers to bear on most of the problems they tackle. (3) Many of the recent AX "expert reasoning" programs are based around inexact plausible reasoning, rather than deduction, and therefore utilize a theorem prover only as one resource, almost as a subroutine, rather than as the central driving mechanism,

2. (1) Complexity of data structures (one working memory consisting of a linear string of tokens, vs. a set of specially-tailored structured DS's). (2) Placement of permanent knowledge (only in rules, vs. distributed between rules and data structures (the knowledge bases)), (3) Complexity of the rules (just a couple of simple operations like pattern-matching and writing a token into memory, vs. the ability to call on arbitrary functions, have side effects, be a meaningful chunk of knowledge to a domain expert). (4) Complexity of the interpreter (simple rule selection schemes, such as cyclic scan, vs. the ability to bring knowledge to bear to choose the best rule to fire next),

3. In all cases, the former program is more driven by tables of low-level knowledge, while the latter is more driven by inferencing off a knowledge base of high-level information. For example, Dragon uses Markov processes to simulate speech at a low level, Chess 4.6 has some of its chess information microcoded into the Cyber; Internist is built around tables of symptom-diseast correlations. The defense of the AI approach comes by way of the following picture:



The conventional non-AI approaches (x) such as microcoding can buy you a hefty linear factor against the combinatorial explosion, but only a linear factor. Ultimately, such programs will not be able to be extended except at an exponentially increasing cost. The current AI programs (o), wallowing LISP behemoths by comparison, arc initially more costly (perform poorer), but ultimately we expect that they have chipped away at the exponent in the problem, that eventually (as machines and problems attacked grow) the curves will cross, and AI programs will perform better, A possible example of this behavior already may be seen with the Dendral program for enumerating structural isomers of a given compound: knowledge of each chemical problem constrains the search through the combinatorial space.

4. The sensorimotor coordination required to walk is far beyond what we can handle now. Thus soccer, and to a lesser extent serving a meal, are quite along ways away. Certain limited forms of cooking, those involving very few motions, will be the first of these to arrive. The more intellectual tasks are certainly bound to precede all of these physical ones. A great deal of thought has gone into arithmetic, and is going on even now with CAI efforts. Thus that may be the first out of the six tasks to be successfully carried out automatically. Planning a meal requires so much less real-world knowledge than teaching Shakespeare that it will come about much sooner. So our ordering is: 1st – teaching arithmetic, 2nd – planning a meal, 3rd/4th – cooking a meal, teaching Shakespeare, 5th – serving a meal, 6th – teaching soccer.

5. "Actors" are modular units of representation, as'developed by Carl Hewitt of MIT, and function by message-passing. They are appropriate to coordinating a large network of simple processes.

"$\alpha\beta$ technique" refers to a tree-pruning procedure for cutting down the amount of nodes necessary to expand when carrying out a minimax search in an AND/OR tree. By comparing the expected value of a branch against (a) the best value you know you can force and (b) the worst value you know you have to settle for, the program can avoid searching many branches. It is usually preferable to a blind minimax search, and is commonly used for evaluating game trees.

"'British Museum algorithm" refers to an exhaustive search, and is relevant only when nothing else is available, or for tiny problems. The name comes from the metaphor of having enough monkeys at typewriters eventually produce all the works in the British Museum.

"Goal-directed search" refers to the problem-solving strategy of working backward from a goal, setting up relevant subgoals, and choosing the next node to expand as one that is necessary for achieving the goal or current subgoal. I$ is generally useful whenever a sense of direction toward the goal is possible.

"Simon's ant" refers to the behavior of an ant crawling on a beach: it appears to follow a very complex path, but when we look closer we see that it was really just avoiding obstructions, that the complexity was in the environment, not in the performer. The point is that simple control mechanisms in a complex environment can produce very complex behavior.

6. (a) *Best* (*S*, *par*) ≡

$$\max_{i,j,k \in S}[par \times Best\ (S \text{ with } S_i\ S_j\ S_k \text{ decremented by } 1, -par)]$$

where S is the list of pile heights, initially S = (11 10 9 8 7 6 5 4 3 2 1); *par* is the parity, which is 1 when, you play, -1 when your opponent plays; and where we assume that the maximal *i*, j, *k* will be bound and available at the end of calling *Best;* thus their final value dictates the initial move, and the final value returned by *Best* is the score (hopefully positive!) we can expect to obtain against a perfect opponent.

(b)

$S \leftarrow$ (11 10 9 8 7 6 5 4 3 2 1)
par ← I
Move ← (0 0 0)

Best(S, par) ≡
. Tempscore ← 0
$\forall S_i \in S$, if $S_i < 0$ then return — 999999999
else if $S_i = 0$ then
Tempscore ← Tempscore — par
Remove S_i from S
if $\sum_i(S_i) < 3$ then
$M = \{i | S_i \in S\}$
Return Tempscore — [par × length(S)]
Move ← *(ijk)* maximizing the quantity
Tempscore + [par × Best(S with $S_i S_j S_k$ decremented, — par)]
which maximal quantity is Returned as the value for this function.

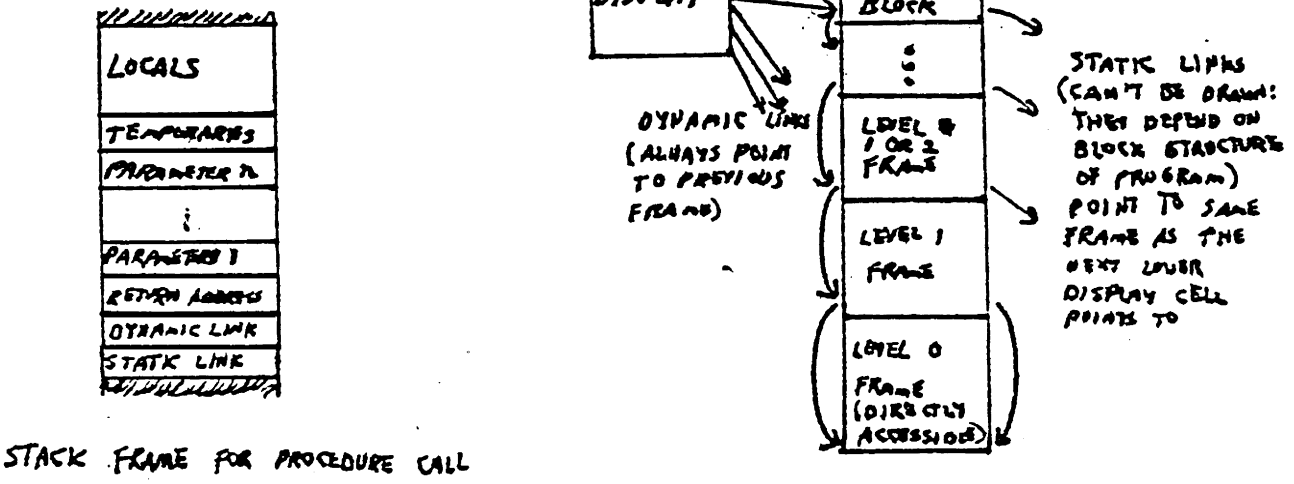
As above, the value of the top-level call of Best will be the expected final score, and the value of Move will be the pile-numbers of the piles from which the three coins should initially be removed.

(c) Lisp comes to mind, not only because this is the AI *section* of the exam, but also because of its ability to handle recursion, list deletions, forall/foreach mappings, etc. In short, translating the (b) program into Lisp would take but a, small fraction of the time it would take for Basic, Cobol, and other straw men.

Systems

1. The following solution works for most general register machines, such as the **PDP-10.** More **details** can be found in Gries, etc.

One register is used for the Stack Pointer; a contiguous block is used for display registers. All of these registers must be **usuable** as index registers on your machine. With a fixed **size** display it is OK to limit the maximum procedure nesting, say to seven levels. No display level is needed for top-level **global** variables, which art directly accessible. Each display register points to the beginning of the local variable **area** of the stack frame for its display level, so that variables on that level may be accessed. When exiting a procedure, its static and dynamic links are . found via the display register for the block **level** of the procedure.



2. There is no need for restrictions on the ordering of declarations, since forward references can be resolved in a **later** pass. .

The presence of separate passes adds modularity to the compiler, in that each pass is concerned with a small part of compilation rather than every part (syntax analysis, semantic analysis, code generation, optimization, etc.) at once.

If code generation is handled in a separate pass, then only this pass need be rewritten in order to transport the compiler to another machine.

Optimization is facilitated because the compiler can always know, through information obtained from an earlier pass, which subexpressions will be needed later, how many registers will be needed later, etc.

Also, the debugging capabilities in modern Lisp languages make it *very easy* to check the partially-complete program, *to* change the sign in *front of* "par" and try it again, etc., compared to compiled languages (and interpretive ones without a "break package").

(d) Modelling the user seems appropriate. We can imagine creating and using a large knowledge base of models for various types of players (neophyte, mathematician, etc.), and trying to quickly ascertain which "stereotype" our current player falls into. Each class would have its own special weaknesses which could be taken advantage of. In addition, a special model could be accreted for each individual who played the system, and it could thereby know and exploit his own weaknesses (e.g., laying a trap which a perfect opponent would ignore).

(e) The results of some of the searches may be stored in a place where they *can* be accessed when later called for again, so as to avoid Fe-computing them. As a simple example, consider the situations where after 4 moves, there were 16 distinct ways to reach the identical state of the piles. It would be a waste to compute n^n where $n!$ will do. Also, there are isomorphs that arise due to the fact that what matters is merely the SET *of* pile heights; thus (1 2 0 4 0 0 4 5 5 9 11) is the same as (1 2 0 0 0 0 5 5 9 4 11). We can imagine storing the results under the SORTED list of pile heights, in this case (0 0 0 0 1 2 4 4 5 5 9 11). After solving this once, the second time we'd have the program check *for* such an entry, it would find and return it immediately, without recomputing it.

Almost the same method is used in both cases: references to yet undefined symbols are kept in a linked list. One usually uses the address fields of the jump instructions to store the link to the previous forward jump to the same address. When the symbol becomes defined, this list is traced and every member of it is corrected to jump to the newly found address. When compilation is in-core, the compiler traces the list. When relocatable code is produced, the linked list ends up on the object file (since the links arc in the address fields of the instructions being generated) followed by a "symbol define" loader command at the proper place; upon encountering this command, the loader defines the symbol and traces the linked list, correcting the instructions it previously loaded.

3. Below is one set of choices. There are many possibilities; grading will be based on the simplicity and consistency of your answer. If your answer does not allow static type determination, it will be penalized, for Pascal is strongly typed. The following solution is that used in Algol 60, modified to allow static type determination.

a integer, b integer: result type is integer;

$b \geq 0$: $a**0 = 1$, $a**(n+1) = a*(a**n)$

It is simpler to make even $0**0 = 1$ rather than ERROR.

$b < 0$: $a = 0$: ERROR;

$a = 1, -1$: $a**(-b)$

$|a| \geq 2$: 0

(since these possibilities are useless, a better solution is simply ERROR for all $b < 0$).

a real, b integer: result type is real;

$b \geq 0$: Exactly as in previous case

$b < 0$: $a = 0$: ERROR, $a \neq 0$: $a**b = 1/(a**(-b))$

a real, b real: result type is real;

$a > 0$: $a**b = exp(b*ln(a))$

$a = 0$: $b > 0$: $a**b = 0.0$

$b \leq 0$: $a**b =$ ERROR

$a < 0$: $a**b =$ ERROR (simpler to call it ERROR whenever $a \leq 0$)

The weight of evidence is against the inclusion of exponentiation. It is rarely used, even in numerical analysis programs. Its rules are complex and hard to remember, especially because of the many differing choices which can be made. It adds other possible confusion to users of the language, e.g. what is the priority of **, and does it associate to the left or to the right? Pascal is such a simple language that the added complexity of this operator would be very noticeable.

A contrary answer may receive full **credit** if it gave an application where this **operator** is essential, and if the exponentiation rules it presented are simple enough. Advocacy of restricted exponentiation, such as integer powers only, may also **receive** credit.' However, merely pointing out that exponentiation is easy to implement is not sufficient justification. Simplicity of the language is more important than ease of implementation; general exponentation is too complex from the user's standpoint. .

4. The following program solves the problem

```
procedure add1(a); a ← a + 1;

procedure foo(a, b);
    x ← 1;
    if a = 1 then comment name or reference;
                if b = 2 then print("name") else print("reference")
    else comment value or value - result;
                add1 (x);
                if x = 2 then print( "value - result") else print( "value");

x ← 0;
foo(x, x + 1);
```

5. The Banker's Algorithm was designed by E.W. Dijkstra for deadlock-free resource management in operating systems. Each process must declare in advance how many units of each resource it may need in order to run; while running each process requests requests and releases units of those resources, staying within its declared limits. If a process makes a request which cannot safely be granted without allowing the possibility of deadlock, then that process can wait; eventually its request will be granted. Each process is required to eventually return all of **every** resource it has borrowed, assuming its requests are granted in a finite time. The algorithm gets its name from the idea of making loans to processes, which **are** later repaid.

6. The program and correctness proof, using semaphores, is **taken** from Brinch-Hansen, Operating *Systems* Principles, page 95, changing every "1" to a "2". The following program works for process *i*, given a global semaphore *mutex*, initially 2 (the only difference from the critical region case, where the initial value is 1):

```
repeat
    wait(mutex);
    serious-region-i;
    signal(mutex);
    non-serious-region-i
forever
```

7. Here is a possible answer. Exact adherence to Simula conventions is not required.

```
class stack(n); integer n;
begin
    integer stackptr;
    integer array stackarr[1:n];
    procedure push(v); integer v;
    begin
        stackptr := stackptr+1;
        stackarr[stackptr] := v
    end push;
    procedure pop(v); integer v;
    begin
        V := stackarr[stackptr];
        stackptr := stackptr- 1
    end pop;
    boolean procedure empty;
    begin
        empty := stackptr = 0
    end empty;
    comment initialize stack to be empty;
    stackptr := 0
end stack;
```

Declarations of stacks:

```
ref(stack) a,b,c
ref(stack) array sa[ 1:20]
```

Creation and initialization of stack instances:

```
a :- new stack(5)
b :- new stack(i*j)
for i := 1 step 1 until 20 do sa[i] :- new stack(i)
```

Stack operations:

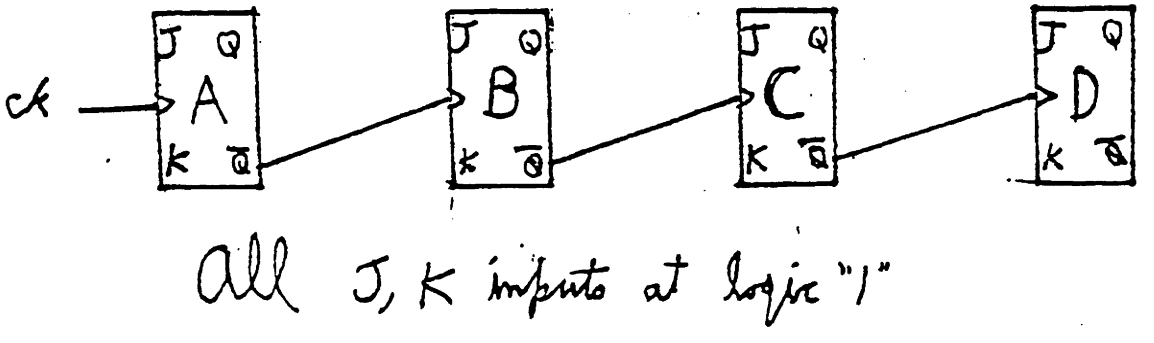
```
a.push(23)
b.pop(k)
if sa[i].empty then sa[i].push(i) else sa[i].pop(j)
```

Hardware

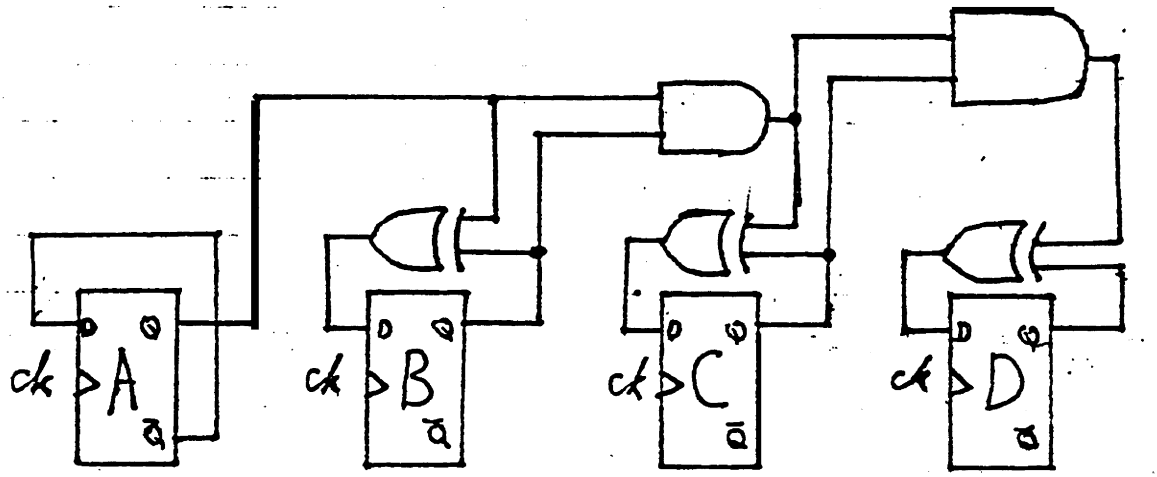
1. (a) This is just a binary up-counter, state **table:**

a	*b*	*c*	*d*
0	*0*	*0*	*0*
1	*0*	*0*	*0*
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	**1**
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

(b) You should have been able to get ZERO gates, because this is the binary ripple counter.



(c) Using K-mapping leads to a mess with this problem; you should know how to make D flip-flops act like T flip-flops as in the solution below:



(d) Possible answers: Likelihood of glitching-is reduced because all flip-flops change state at nearly the same instance; they keep the circuit conceptually simple because only a single clock is used; the counter does not go through "intermediate states" when counting, because all transitions are simultaneous; the time required to change states is that of a single flip-flop, no matter how many the counter contains.

(e) Possible answers: On an IC, the circuit requiring fewer gates often requires more chip area than another circuit (the rule for ICs is: minimize wires, not gates). The larger circuit may be faster, since minimizing gates often requires that gates be cascaded, increasing the delay time. The larger circuit may involve prepackaged MSI or LSI chips, and thereby be cheaper and simpler than smaller circuits not taking advantage of prepackaged devices. The larger circuit may be simpler for humans to understand, debug, etc. (many "structured programming" ideas apply to hardware too).

2. RTL Resistor Transistor Logic – obsolete, slow, high power consumption; used in early ICs

MOS Metal Oxide Semiconductor – slow, low power, high/very high density, used for large memories, microprocessors, other LSI

ECL Emitter Coupled Logic – very fast, costly, high power, difficult to design with, used for cache memories, high performance CPUs

TTL Transistor Transistor Logic – fast, fairly high *power,* cheap, easy to design with, commonly used in many applications

Schottky A faster, more expensive TTL

Josephson Junction Experimental ultra-fast logic (picosecond switching speeds) based on superconductivity

IIL Integrated Injection Logic – fast, high density, fairly high power, rarely. used; in some ways a refinement of RTL

3. (a) What is a stack machine? A machine, such as the B6700, in which a stack is maintained by the hardware. Most instructions take their operands from the stack, and hence have no address fields. There are no general registers; all computation is done on the stack.

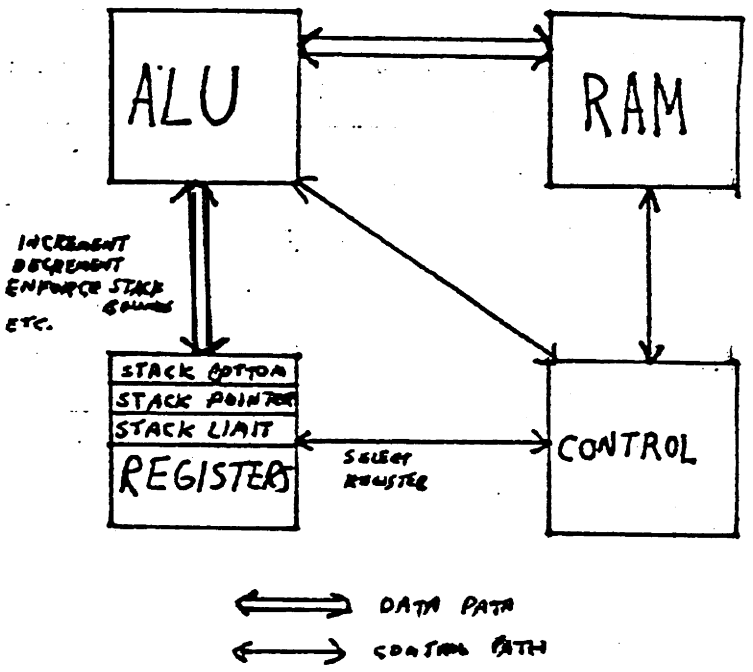
(a) What is a register machine? A machine, such as the PDP-10, IBM 360, ad infinitum, which provides an array of registers for general use. Instructions can address either registers, memory, or both. Stack manipulation instructions may be available, but their use is optional.

(c) What are each's advantages over the other? Stack machines, by eliminating registers, also eliminate the thorny problem of register allocation; also, there are no registers to save and restore around procedure calls. Code generation for expressions is greatly eased, for stack operations always put intermediate results exactly where needed for continued expression evaluation. By eliminating register operands, the number of addressing modes is reduced, simplifying the instruction set. Since most instructions lack address fields, code can be very compact.

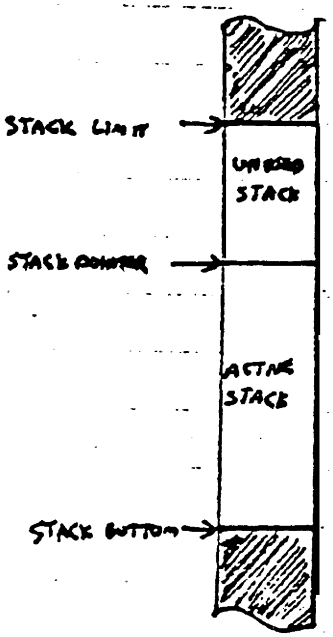
Register machines are usually faster than stack machines, for they lack the overhead required to maintain the stack, and they are able to use high speed logic for the registers (whereas the stack must be kept in slower main memory). Clever programmers using registers to storeoften-accessed variables can produce programs which run much faster than possible on stack machines. Register machines are more appropriate than stack machine6 for the commonly used languages FORTRAN, BASIC, COBOL.

The main idea is that the stack is kept in main memory, using a special register for the stack pointer; other designs are possible.

Diagram for 3(d):



Stack organization in RAM



4. M. **Wilkes** -Invented subroutine libraries; wrote first programming text; built EDSAC

T. **Kilburn** – Invented index registers, virtual memory

S. Gray – Designer of the world's fastest computers: **CDC** 7600, **Cray-1, Cray-2**

G. **Amdahl** – Major figure in development of IBM 360; later started own company, marketing a fast and relatively cheap copy of the 370 (called the 470 – what else?)

G. Eell – **Designed PDP-11;** co-authored Computer Structures: Readings and *Examples.*

1. *(a)*

```
t ← root;
while t ≠ Λ do begin
    if left(t) = Λ then begin
        visit(t); t ← [ right(t) ]
    end
    else begin
        p ← left(t);
        comment p is a temporary variable used only in this block;
        while right(p) ≠ Λ and right(p) ≠ [ t ]
            do p ← right(p);
        if right(p) = Λ then begin
            comment modify tree link to "remember our place";
            right(p) ← t; t ← left(t)
        end
        else begin
            visit(t);
            comment fix up tree link;
            right(p) ← [ Λ ]; t ← [ right(t) ]
        end
    end
end;
```


(b) Simply **move** the *visit(t)* statement in the last **else** clause up to the beginning of the corresponding **then** clause.

2. (a) Use a doubly-linked list with the numbers in sorted order, and keep pointers to the first, last, and current median elements of the list. We also need a bit to remember if the number of elements in the list is odd or even in order to update the median pointer correctly. All operations can then be done in constant time.

(b) Use a binary search tree. All operations take O(log n) time, on the average. To ensure $O(\log n)$ worst-case time, you must use one of the varieties of balanced trees, such as AVL trees or 2-3 trees.

(c) Use an array of size 50, with the ith entry pointing to a sorted list of **all** jars with capacities $\in [i, i+1)$. Insert takes about $n/100$ steps, on the **average,** while find-delete takes constant time. For very large n (say $n > 2000$), it would be better to use some type of priority queue for each of the sublists. This **reduces** the time to $O(\log n)$.

It should be noted that we can also use the method of part (c) for part (b), and vice-versa.

(d) Use an array of size 50, with the ith entry pointing to the list of all jars with capacity i. All operations take O(1) time.

3. (a) The minimum number of registers required to compute expression e is $f(e)$, where

$$f(\text{variable}) = 0$$

$$f(e_1 \circ e_2) = \begin{cases} \max(f(e_1), f(e_2)), & \text{if } f(e_1) \neq f(e_2); \\ f(e_1) + 1, & \text{if } f(e_1) = f(e_2). \end{cases}$$

For example, $f(((a \circ b) \circ c) \circ (d \circ ((e \circ f) \circ g))) = 2$.

The order in which the operations should be performed to achieve this minimum number is recursively defined as follows: For each expression $e_1 \circ e_2$, if $f(e_1) \geq f(e_2)$, then compute the left operand e_1 first. Otherwise compute the right operand first.

(b) The shortest expression with $f(e) = k + 1$ has the form $e_1 \circ e_2$, where e_1 and e_2 are the shortest with $f(e_1) = f(e_2) = k$. Thus by induction on k the minimum number of operators, $g(k)$, satisfies $g(0) = 1$, $g(k) = 1 + 2g(k - 1)$; the solution is $g(k) = 2^{k+1} - 1$.

(1a,b) Good-good;bad-good. If the data x_i are spaced a distance h apart, then a perturbation of one value $f(x_i)$ by a quantity ε will affect any estimate of the integral of f or the derivative of f at x_i by $O(\varepsilon h)$ or $O(\varepsilon/h)$, respectively. These results are inherent in the two problems; Simpson's rule and finite differencesdo not introduce avoidable errors.

(1c) Bad-good, for the problem as a whole. For isolating just the roots 0 and 2, the answer would be good-good. For the three multiplicities of the root x=1, however, the problem is ill-conditioned: a change in the coefficients of magnitude $O(\varepsilon)$ may correspond to a change in these three roots of magnitude $O(\varepsilon^{1/3})$.

(1d,e) Good-bad,good-good. The inverse of the matrix $A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 2 \end{bmatrix}$ is $\begin{bmatrix} -2 & 1 \\ 1 & -\varepsilon \end{bmatrix} + o(e)$. . This does not change too much when A is changed slightly, so the problem is well-conditioned. Gaussian elimination with partial pivoting solves it stably; Gaussian elimination without partial pivoting, however, is unstable because it involves a multiplication by $1/\varepsilon$, which amplifies rounding errors.

(1f,g) Bad-good,bad-good. This matrix is nearly singular, so the problem is ill-conditioned. For positive definite matrices, Gaussian elimination is a stable algorithm, even with no pivoting.

(2a) Let us write

$$\alpha f(x) = \alpha f(x)$$
$$\beta f(x+h) = \beta[f(x) + hf'(x) + h^2 f''(x)/2 + \ldots]$$
$$\gamma f(x+2h) = \gamma[f(x) + 2hf'(x) + 2h^2 f''(x) + \ldots]$$

We wish to find values for α, β and γ so that the sum of these terms equals $0f(x) + 1f'(x) + 0f''(x) + O(h^k)$, where k is as high as possible. Since we have three parameters, we can in effect specify the f(x), f'(x), and f''(x) coefficients in the sum. These should be 0, 1, and 0. Thus we must solve

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & h & 2h \\ 0 & h^2/2 & 2h^2 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The solution to this system turns out to be $(\alpha,\beta,\gamma) = \frac{1}{h}(-\frac{3}{2}, 2, -\frac{1}{2})$.
NOTE: An alternative approach would be to derive a formula that is exact for the monomials $f(x) = 1$, x, and x^2.

(2b) A satisfactory, but not completely rigorous, approach would be to apply the formula derived in part (a) to $f(x) = x^3$. Instead, let us observe that the $f'''(x)$ term in part (a) will dominate the error. Applying a mean value theorem, we *may* write

$$f(x+h) = f(x) + hf'(x) + h^2f''(x)/2 + h^3f'''(\xi_1)/6$$
$$f(x+2h) = f(x) + 2hf'(x) + 2h^2f''(x) + 4h^3f'''(\xi_2)/3,$$

where $\xi_1 \in [x,x+h]$ and $\xi_2 \in [x,x+2h]$. We now compute

$$\alpha f(x) + \beta f(x+h) + \gamma f(x+2h)$$
$$= f'(x) + h^2f'''(\xi_1)/3 - 2h^2f'''(\xi_2)/3.$$

Assuming that $f'''(\xi)$ is continuous on $[x,x+2h]$, the latter two terms may be averaged, yielding

$$f'(x) - h^2f'''(\xi_3)$$

for some $\xi_3 \in [x,x+2h]$. So we have *at* last

$$\left| y' - f'(x) \right| \leq \frac{1}{3} h^2 \max_{\xi \in [x,x+2h]} \left| f'''(\xi) \right| .$$

(2c) $|\sin'''(x)| \leq 1$ for all x, so we may require $(1/3)\,h^2 \leq 0.00005$.
That is, $h \leq \sqrt{0.00015} \approx 0.012$

(3a) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, for example.

(3b) For $k = 1,\ldots,n-1$, the submatrix formed from the first k rows and columns of A should be nonsingular.

(4) Even $\frac{0}{0}$ will probably trigger a divide-by-zero message, so the computation of F is a red herring. The question is, in what base(s) b will E come out *exactly* 0? The H := 1./2. computation will be exact provided b is a multiple of 2, and X := 2./3. - H will be *exact* provided b is a multiple of 3. So a sufficient answer is <u>b = 6</u> or any multiple thereof. (Additional possibilities exist, in which the rounding error would cancel in the computation of E.)

(5a) $\|x\|_\infty$ — i.e., $\max_i |x_i|$.

(5b) For $p<1$, $\| \ \|_p$ fails to satisfy the triangle inequality, and hence is not a norm.

(5c) For any $p > 1$ the quantity $\sum_i |x_i|^p$ may be very large or very small compared to $|x_i|$ or $\|x\|_p$. A direct implementation of the definition risks unnecessary overflows and underflows.

Computer Science Department - Comprehensive Exam Programming Project

Winter 1979 -- Thursday, January 4, 1:00p.m. to Tuesday, January 9, noon.

The object of this problem is to prepare an interpreter for the "linear equation language" described below, This language defines the value of variables implicitly by means of linear equations, instead of explicitly by means of assignment statements. Your implementation should be on-line, i.e., interactive with the user.

Note. Your interpreter should be written in well-structured- easy-to- understand code. It may be written in any "ALGOL-like" language (including ALGOL W, SAIL, PASCAL) or in your favorite dialect of LISP. Other languages may be used but only by special arrangement with the committee.

Here is the syntax for the language (which incidentally is called LEELAND, for "Linear Equation Language Allowing Nonexplicit Definitions"):

(variable) ← A|B| . . . |Z|a|b| . . . |z (52 variables in all)

(digit) ← 0|1|2|3|4|5|6|7|8|9

(digit string) ← ⟨digit⟩|⟨digit string⟩(digit)

(constant) ← (digit string⟩|⟨digit string).(digit string⟩

⟨primary⟩ ← (variable)\{constant⟩|(⟨expression⟩)

⟨term⟩ ← ⟨primary⟩|⟨term⟩⟨primary⟩|⟨term⟩/⟨primary⟩

(expression) ← ⟨term⟩|+⟨term⟩|-⟨term⟩|⟨expression⟩+⟨term⟩ |⟨expression⟩-⟨term⟩

(equation) ← (expression) = ⟨expression⟩|⟨equation⟩ = (expression)

(print statement) ← (expression):

(statement) ← ⟨equation⟩⟨cr⟩|⟨print statement⟩⟨cr⟩

Examples :

(variable)	x
(digit}	1
(digit string)	14
(constant)	3.14
⟨primary ⟩	x
⟨term⟩	xy/z
(expression)	xy/z - 3.1'1
(equation)	xy/z - 3.14 = x = w
(print statement)	x:
(statement)	x: (cr)

Here (cr) stands for the "carriage return" character, All blank spaces and other characters not appearing in the above syntax are ignored. The syntax is ambiguous with respect to constants: For example, the ⟨term⟩ 3.14 can be regarded either as a (constant} or as ⟨term⟩⟨primary⟩ where the ⟨term⟩ is 3.1 and the ⟨primary⟩ is 4 . This ambiguity is resolved by the further rule that a ⟨constant⟩ may not be preceded or followed by a (digit) .

Expressions have their normal-meaning in mathematics; for example, ⟨term⟩⟨primary⟩ stands for the value of the (term) times the value of the (primary) .

Note that each (statement) ends with a (cr) . The user of LELAND, when prompted, types a (statement) ; the interpreter processes it and prompts the user for another, repeating this cycle until the user gets tired and stops the job. If an erroneous statement is typed, LELAND gives a helpful description of the error and stops further evaluation of that statement; the user will be prompted to try again as if the offending statement had not occurred. (However, in an equation statement of the form $\alpha = \beta = y$ where " $\alpha = \beta$ " is OK but y is erroneous, the " $\alpha = \beta$ " equation will be accepted by LELAND.)

Initially all variables have undefined values, but each new equation defines one of the variables (perhaps in terms of others). For example, after the three statements

x+y = 2

x-y = z

x:

LELAND will print " 1+ .5z " indicating that the value of x must be one more than half the value of z , based on the equations given so far. If the next equation is " 2x = 3z ", the interpreter will know that x = 1.5 , y = .5 , z = 1 .

In order to do this in a reasonably simple way, LELAND allows multiplication only when at least one of the two operands being multiplied has a known value based on previous equations; similarly, division is allowed only when the value of the divisor is known and non-zero. Thus, LELAND would complain if the three statements above were followed by the equation " xy = z "; but " xy = z " would be legal if either x or y had a known value.

All this is accomplished as follows: Inside LELAND, each variable is considered to be either "independent" or "dependent". Initially all variables are independent, but each valid equation makes another variable dependent. Once a variable becomes dependent, it never becomes independent again. A dependent variable is represented internally as a linear combination of independent variables; in other words,

$$D = c_0 + c_1 I_1 + \cdots + c_k I_k$$

where the c's are floating-point constants, the I's are variables that are currently independent, and D is the dependent variable being represented. If k = 0 , variable D is said to be "known".

The stated restrictions on multiplication and division ensure that LELAND can reduce any new equation $\alpha = \beta$ to the form

$$c_0 + c_1 I_1 + \cdots + c_k I_k = 0$$

where the c's are floating-point constants with $c_j \neq 0$ for $j \neq 0$, and where the I's are variables that are currently independent. If k = , the

equation is either redundant ($c_0 = 0$) or inconsistent ($c_0 \neq 0$), and the user is given an appropriate message. If $k>0$, a coefficient c_j with largest absolute value is selected and variable Ij changes from independent to dependent. Its current value will be

$$Ij = -c_0/c_j + \Sigma_{i \neq j} (-c_i/c_j) I_i ,$$

and the values of any other dependent variables that currently involve I_j will be simplified in accordance with this new value.

For example, after the equation " x+y = 2 " above, LELAND will first obtain

$$-2 + x + y = 0$$

and then either x or y will become dependent. (LELAND is free to decide which.) Say y becomes dependent, so that y = 2-x ; then the second equation x-y = z will reduce to

$$-2+2x-z = 0 ,$$

hence x will become dependent and equal to 1+ .5z . On the other hand, if LELAND had decided to make x dependent instead of y after the first equation, the second equation would have reduced to

$$2 - 2y - z = 0 ,$$

making y dependent and equal to 1 - .5z . This new dependency would also be reflected in the current value of x , which would change from 2-y to 1+ .5z .

When LELAND forms linear combinations of coefficients, the floating-point quantity x+y is always replaced by zero whenever $|x+y| < 0.00001 \max(|x|, |y|)$. A similar thing happens in floating-point subtraction. This gets around problems caused by rounding errors, (For example, 3 times 1/3 isn't exactly 1 in floating-point arithmetic, so LELAND might not otherwise realize that x is known after the equations

$$x = 3z + y + 1$$

$$z = -Y/3$$

have appeared.)

Your job is to implement such a system. Hand in well-documented code, together with examples of test runs that demonstrate its correct working in well-chosen, nontrivial cases. Be sure to devise a good way to indicate syntactic and semantic errors to the user. Your program should be reasonably efficient in its use of time and space.

Algorithms and Data Structures

1. Searching (24 points)

Mr. J. H. Quick (a student) needs to search an ordered table

$$A[1] < A[2] < \cdots < A[n]$$

to find the largest j such that $A[j] \leq x$; and if $x < A[1]$ he wants j to be zero. But he doesn't wish to use binary search, he prefers to use the following scheme (depending on an integer parameter h):

```
j := 0;
while j + h ≤ n and A[j + h] ≤ x do j := j + h;
while j + 1 ≤ n and A[j + 1] ≤ x do j := j + 1.
```

Fortunately he is using a compiler that will not compare $A[j + h]$ with x when the test "$j + h \leq n$" fails; so this program will work, for all positive integers h. He tried it with $h = 10$, but he wonders if there is a better value.

(a) Let $f_1(h, j, n)$, $f_2(h, j, n)$, $f_3(h, j, n)$, $f_4(h, j, n)$, $f_5(h, j, n)$, and $f_6(h, j, n)$ be the number of times Quick's program evaluates "$j + h \leq n$", "$A[j + h] \leq x$", "$j := j + h$", "$j + 1 \leq n$", "$A[j + 1] \leq x$", and "$j := j + 1$", respectively, as a function of the given positive integers h and n and the final value of j. For example, if $h \leq$ n and j = 0 we have $f_1 = f_2 = f_4 = f_5 = 1$ and $f_3 = f_6 = 0$. Express these six functions in terms of the quantities $\lfloor j/h \rfloor$, $\lfloor n/h \rfloor$, j mod h, and nmod h.

(b) Given n and h, determine the worst case of the algorithm, assuming that the total running time is

$$f(h, j, n) = 1 + \sum_{i=1}^{6} f_i(h, j, n).$$

In other words, determine the value of j that maximizes $f(h, j, n)$.

(c) When $n = 80$, what is the best choice of h, in the sense that the worst case running time is minimum?

(d) For large n explain how to choose an optimum h, and give an approximate formula for the worst-case running time as a function of n when the best h has been selected.

2. Height and depth in **binary trees** (20 points)

An extended binary tree **is a** binary tree in which all nodes are either "internal" (branch nodes), having two sons, or "external" (leaf nodes), having no sons. At **each** node x of an extended binary tree let h(s) be the length of the longest downward path from x to **a** leaf, and let d(x) be the length of the shortest **downward** path **from x to a leaf.** Thus, if x **is a** leaf we have h(x) = d(x) = 0, while **if x is a branch node having sons $l(x)$** and t(x) we have

$$h(x) = 1 + \max(h(l(x)), h(r(x))),$$
$$d(x) = 1 + \min(\mathrm{d(l(x))}, d(r(x))).$$

(a) Draw an extended binary tree having $d(\text{root}) = 4$, where the **total number of** nodes **is** *as* **small as possible.**

(b) A *height-balanced* tree **satisifies** $|h(l(x)) - h(r(x))| \leq 1$ for all branch nodes x. **Prove that** $h(x) \leq 2d(\mathrm{x})$ **for** all nodes x in a height-balanced tree.

(c) Draw an extended binary tree having $h(\text{root}) = 7$ **and** with $h(x) \leq 2d(x)$ for all nodes x. **Your tree** should. have the smallest possible total **number of** nodes subject to these conditions, (It need not be height-balanced.)

(d) Prove **or** disprove: In an extended binary tree having $h(x) \leq 2d(x)$ for all nodes x, the height **of** the **root is** $O(\log n)$, where n is the **total** number of nodes.

3. Algorithm design (16 points)

Suppose we **are** given a box of a specified capacity and a collection of objects of **various sizes,** and we wish to pack the box as fully as possible. **More** formally, if the objects have sizes $\{x_1, x_2, \ldots, \mathrm{x},\}$, then we want to find a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} x_i$ is **as** large as possible, but not greater than the capacity c of the box.

Assuming that the x_i are positive integers, design an algorithm to find an optimal packing. Write your algorithm as a Pidgin-Algol program or in Knuthian style. Say why you think your algorithm **is** good, giving estimates of its running time and memory **requirements.** To facilitate practical **considerations, assume** that $c \leq 1000$ and $n \leq 100$.

Note: Your program must output not only the size of an optimal packing, but **also** a subset S of the objects such that S achieves this maximum.

1. Howard Cosell (60 points)

While the social issues raised by automation are important and difficult, few could dispute the goal of creating an AI program to replace Howard Cosell. This question deals with the design of such a system: a computer program capable of following the action of a professional sports event, analyzing it, reporting its analyses, and leaving no silent moments.

(a) [10 points] Basedon current AX work, what aspects of a sport would you expect to make this problem more/less difficult? Cite specific aspects of specific AI programs where appropriate in your discussion. Based on this, select an appropriate sport that you will consider in the remainder of the problem. (If you truly are unfamiliar with all professional sports, you may choose the task of commentator for a live chess match.)

(b) (20 points] What kinds of information could be used by such a program, to enable or merely to facilitate its operation? For each type, indicate an appropriate representation (and/or data structure), a rough estimate of the amount of information desirable, the difficulty *of* obtaining it, and its value to the program.

(c) [15 points] Sketch the flow of control through the program, Note how each type of knowledge mentioned in part (b)above is accessed and used. (If you sup plied alternatives for representing some type of information in (b), then choose one of them here.)

(d) [10 points] What are the pros and cons of taking such a knowledge-based approach? Consider, for example, changing the program to another sport, time and space costs, debugging the system initially, etc.

(e) [5 points] Assume that you *were* going to spend about two years working on this project (e.g. as a thesis). Which of the information sources you mentioned in part (b) (and control structures you sketched in part (c)) would you include, and which would you exclude? Explain your choices.

Systems

Problem 1. (5 points)

Give some comparative advantages and disadvantages of the following parsing methods:

precedence

LL

LR

LALR

Problem 2. (5 points)

Given a directed acyclic graph representation of a basic block. program fragment, what limitations must be imposed on the order of code generation?

Problem 3. (5 points)

What is aliasing (when this term is applied to programming languages)? Why is it considered bad? What additional rules could you add to a language like Pascal to reduce aliasing?

Problem 4. (10 points)

Define a debugging compiler as one that allows the programmer to change his program during testing without incurring the trouble *or* expense of complete recompilation. Discuss a possible implementation of such a compiler. What types of changes could you support? What information will you need at run time? How does your scheme interact with the code optimization parts of the compiler?

Problem 5. (5 points)

Define binding time. Give an example of early and late binding of some attribute. What are advantages and disadvantages of early and late binding?

Problem 6. (10 points)

Comment on the following proposal for a *new* program verification system:

"It often happens that one version of an algorithm is easier to prove correct than another version. Instead of devoting effort to verifying the difficult program, we should verify the easier program and write a program to show that the two programs are, in actuality, functionally equivalent. The project is to write a general purpose program to detect functional equivalence of any two given programs."

Problem 7. (5 points)

Pascal binds the else clause in if-then-else statements to the most recent then clause. Write an unambiguous, context-free grammar that enforces this binding in the state generating rules,

For simplicity, assume that you are dealing only with assignment statement6 and conditional statements. Thus, the problem is to convert the ambiguous BNF specification

<statement> ::= <assignment> | if <expression> then <statement> |
if <expression> then <statement> else <statement>

into an unambiguous grammar.

Problem 8. (5 points)

How is synchronization of concurrent processes normally done in a message oriented operating system?

Problem 9. (10 points)

Activation record retention is a general programming language scheme that can be used to solve the funarg problem in Lisp and the call by name problem in Algol. Describe briefly how activation record retention can be used as a major element in the implementation of an operating system that is based on monitors and is procedure oriented.

Numerical Analysis

1. Nonlinear equations (15 points)

(a) Suppose we are given the equation

$$x - \sinh x + 3 = 0\,, \quad \text{sinhx} \equiv \frac{e^x - e^{-x}}{2}$$

and propose to solve **it** by a method of successive substitution **in the following manner:**

$$x^{(0)} := \textit{some}\ \text{initial guess}$$

$$x^{(k+1)} := \sinh(x^{(k)}) - 3\,, \quad k = 0, 1, 2, \ldots .$$

Show that this procedure cannot be expected to converge **to** *a* solution **of** the given equation.

(b) How might we modify the procedure so as to achieve rapid convergence to a **correct** solution, still **using** a form of successive substitution?

2. Well-conditioned problems; stable algorithms (25 points)

Given as data the coordinates (u_1, u_2) and (v_1, v_2) of two vectors u and v in the plane, our object is to devise a stable algorithm for computing the angle φ between u and v.

(a) The angle φ can be defined by the formula

$$\cos\varphi := \frac{u_1 v_1 + u_2 v_2}{(u_1^2 + u_2^2)^{1/2}\,(v_1^2 + v_2^2)^{1/2}} \equiv \frac{u^T v}{\|u\|_2\,\|v\|_2}.$$

from which φ might be computed by taking the inverse cosine. We say in general that a numerical problem is well-conditioned if a small change in the data defining the problem results in only a small change in the solution. Show by means **of** this formula that the quantity φ is well-conditioned as a function of the data u_1, u_2, v_1, v_2. *(Hint:* Take the partial derivative of $\cos\varphi$ with respect to u_1, and then from this **compute** the partial of φ with respect to u_1. By symmetry, u_2, v_1, and v_2 are essentially the same.)

(b) We say in general that a numerical algorithm is *stable* if it does not introduce large errors in the computed solutions to problems which are well-conditioned. Show that computing φ by means of the above formula and the inverse cosine is nof numerically stable.

(c) An alternative algorithm is the following. First normalize the vectors,

$$\tilde{u} := \frac{u}{\|u\|_2}, \qquad \tilde{v} := \frac{v}{\|v\|_2},$$

then compute $\alpha := \|\tilde{u} - \tilde{v}\|_2$, $\beta := \|\tilde{u} + \tilde{v}\|_2$. Now, compute

$$\varphi := \begin{cases} \arctan(\alpha/\beta), & \alpha \leq \beta \\ \pi - 2\arctan(\beta/\alpha), & \alpha > \beta. \end{cases}$$

Why is this algorithm better than the one proposed in (b)?

3. Linear least squares and band matrices (20 points)

The vector x that minimizes $\|b - Ax\|_2$ satisfies the normal equations $A^TAx = A^Tb$. Suppose that A is $m \times n$, where $m > n$, and has the property that for $i = 1, 2, \ldots, m$ we have

$$a_{ij} \neq 0 \text{ and } a_{ik} \neq 0 \quad \Rightarrow \quad |j - k| < w,$$

for some positive integer w. The matrix A is then said to be a *rectangular band matrix* with band width w. (For this concept to be useful we should have w $\ll$ n.)

(a) Show that A^TA is a symmetric band matrix and determine its band width.

(b) Show that A^TA is positive semidefinite, and positive definite if A is nonsingular. Assuming A^TA is positive definite, what significance will this fact have when we need to compute the $LL^T (= LU)$ decompositon of A^TA?

(c) If $m \gg n$ and you are given A a row at a time, how would you form A^TA?

(d) Estimate the total number of multiplications needed to form A^TA and to compute the lower triangular matrix L in the factorization $A^TA = LL^T$. It suffices *to* give the leading term.

Theory of Computation

1. Decidability (5 points)

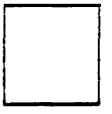
Prove that in languages like Algol and Pascal, it is undecidable whether or not a variable has been assigned a value before it is used,

2. Grammar6 (15 points)

In each of the following three problems, try to give *a* grammar at the lowest level in Chomsky's hierarchy (regular, context-free, context-sensitive, recursive). You need not prove that your grammar is at the lowest possible level.

(a) [4 points] Suppose we want to describe a path consisting of straight segments of length 1 inch, each of which is running in a northerly, southerly, easterly, or westerly direction. Consider the alphabet A = $\{r, l\}$, where r means turn right, l means turn left, followed by a move of 1 inch in the new direction. A string over A specifies a sequence of right and left turns and corresponding moves. Give a grammar generating all those strings over A that describe paths ending in the same direction as they start; e.g. $l\,r$ and $l\,l\,l\,l$ are elements of this language.

(b) [4 points] Consider the alphabet W = {n, s, e, w}, where the elements denote a move of 1 inch in a northerly, southerly, easterly, or westerly direction. Give a grammar *over* W that decribes the set of all moves ending in the starting point. For example, the path



can be described by the strings $n\,e\,s$ w, $e\,n$ w s, and several others for other starting points. Also note that $n\,s$ is a valid path ending in the starting point.

(c) [7 points] Use t he result *of* part (b) and specify a grammar for the language of all closed paths in terms of $A = \{r, l\}$, with the meaning of r and l as in part (a). Each path should end in its starting direction, Of course, not all paths in the language in part (b) have a counterpart in terms of r and l; for example, $n\,s$ doesn't. The path need not be simple; for example, the string $r\,r\,r\,r\,r\,r\,r\,r$ is one legitimate way to specify a square path.

3. Primitive Recursion (10 points)

A function over the non-negative integers is primitive recursive if it can be defined from 0 (zero), the successor function $+1$, and the projection functions $U_m^n(x_1, \ldots, x_n) = x_m$ by function composition and the following recursion schema:

$$f(0, y_1, \ldots, y_n) = g(y_1, \ldots, y_n)$$
$$f(x + 1, y_1, \ldots, y_n) = h(x, y_1, \ldots, y_n, f(x, y_1, \ldots, y_n))$$

where g and h are primitive recursive. For example, the addition function $\text{plus}(x, y)$ is primitive recursive since it can be defined as follows:

$$\text{plus}(0, y) = U_1^1(y), \qquad \text{plus}(x + 1, y) = \text{plus}(x, y) + 1$$

The Ackerman function

$$a(0, y) = y + 1$$
$$a(x + 1, 0) = a(x, 1)$$
$$a(x + 1, y + 1) = a(x, a(x + 1, y))$$

is known to be *not* primitive recursive. Prove, however, that for any fixed n the function $a_n(y) = a(n, y)$ is primitive recursive.

4. NP-Completeness (10 points)

Prove that the following problem is NP-hard:

Given a context-free grammar over the terminal alphabet $\{x_1, \ldots, x_n\}$, does the language defined by the grammar include a string that contains each letter x_j exactly once?

Hint: Use the fact that the problem of determining the existence of directed Hamiltonian circuits in a graph is NP-complete.

5. Demand paging (20 points)

Consider a program on a virtual machine capable of storing b "pages" of a fixed size in its high-speed memory, While the program is running it references a sequence of pages given by the "page trace"

$$p_1 p_2 \cdots p_n .$$

Suppose S_j is the set of b pages in high-speed memory just after p_j is referenced; we need $p_j \in S_j$. If $p_j \notin S_{j-1}$, a "page fault" occurs; some page q_j in S_{j-1} is "pulled" and we have $S_j = S_{j-1} - \{q_j\} + \{p_j\}$, exchanging p_j for q_j. It is convenient to assume that the program starts out with a set S_0 of b completely null pages, and that $q_j = p_j$ when there is no page fault.

It is of interest to consider the best possible sequence of page pulls (the sequence that minimizes the total number of page faults) for a given page trace $p_1 p_2 \ldots p_n$, even though it may be impossible actually to achieve this optimum sequence in practice because it may require knowing the future page requests $p_{j+1} \ldots p_n$ at the time q_j must be chosen.

(a) [15 points] Thc purpose of this problem is to give a constructive proof that an optimum strategy is obtained by the following rule: "When $p_j \notin S_{j-1}$, let q_j be an element of S_{j-1} that does not appear in $\{p_{j+1}, \ldots, p_n\}$, if possible. Otherwise (i.e., if all elements of S_{j-1} occur again), let q_j be the element whose first occurrence in $p_{j+1} \ldots p_n$ is after all the other pages of S_{j-1} have occurred."

The proof can be obtained by repeatedly applying the following idea: "Given a page trace $p_1 p_2 \ldots p_n$ and a corresponding sequence of page pulls $q_1 q_2 \ldots q_n$ such that, for some j with $p_j \neq q_j$, there is an element $q \in S_{j-1}$ and an index $k > j$ such that q does not appear in $p_{j+1} \ldots p_k$ but $q_j = p_k$, then there is another sequence of page pulls $q'_1 q'_2 \ldots q'_n$ such that $q'_1 \ldots q'_{j-1} = q_1 \ldots q_{j-1}$ and $q'_j = q$ and $q'_1 q'_2 \ldots q'_n$ has no more page faults than $q_1 q_2 \ldots q_n$."

The required sequence $q'_1 q'_2 \ldots q'_n$ can be constructed in the following way: Let m bc as large as possible such that q does not appear in $q_{j+1} \ldots q_m$ or in $p_{j+1} \ldots p_m$. Let $r_j = q_j$, and for j $< i \leq m$ let

$$q'_i = p_i, \quad r_i = q_i, \qquad \text{if } p_i = r_{i-1};$$
$$q'_i = q_i, \quad r_i = r_{i-1}, \quad \text{otherwise.}$$

Finally if $m < n$ let $q'_{m+1} = r_m$, and let $q'_i = q_i$ for $m + 1 < i \leq n$.

Prove that $S'_i = S_i - \{q\} + \{r_i\}$ for j $\leq i \leq$ m and $S'_i = S_i$ for all $i > m$. And prove that the sequence of page pulls $q'_1 q'_2 \ldots q'_n$ leads to no more page faults than $q_1 q_2 \ldots q_n$ does.

(b) [5 points] One of the page pulling algorithms often used in practice is the so-called "least recently used" rule: If $p_j \notin S_{j-1}$, the page q_j that is pulled is a null page if any null pages are present, otherwise q_j is the page whose last occurrence in $p_1 \ldots p_{j-1}$ comes before occurrences of all the other pages *in* S_{j-1}.

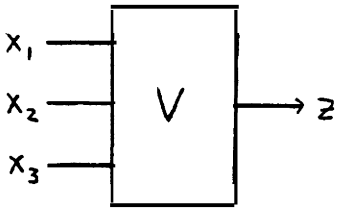
Construct a page trace scheme for which this rule leads to about b times as many page faults as the optimum strategy does.

Hardware

Problem 1. (30 points)

Using AND, OR, NOT, and XOR gates,

(a) Design a combinational circuit V such that $z = 1$ if and only if at least two inputs are equal to 1.



Write the Karnaugh map for this function, and write a logical expression for V.

(b) Design a synchronous sequential circuit that will have its output become and remain equal to 1 only after 3 successive disagreements have occurred between x_1 and z. State what assumptions you make about the clocks you use in your design.

Problem 2. (15 points)

(a) State the conditions that determine the occurrence of an overflow when adding two 2's complement numbers.

(b) Give an example of such an addition using 8-bit numbers.

Problem 3. (15 points)

TTL circuits can have three different output configurations. List each of them, defining their properties and discussing advantages and disadvantages of each.

Algorithms and Data Structures

1. (a) Clearly $f_3 = \lfloor j/h \rfloor$ and $f_6 = j \bmod h$, since j starts at zero. By Kirchhoff's law we have $f_1 = f_3 + 1$ and $f_4 = f_6 + 1$. Furthermore we have $f_2 = f_1$ except that $f_2 = f_1 - 1$ when $\lfloor j/h \rfloor = \lfloor n/h \rfloor$; similarly $f_5 = f_4$ except that $f_5 = f_4 - 1$ when $j = n$.

(b) $f = 3 + 3\lfloor j/h \rfloor + 3(j \bmod h) + (\lfloor j/h \rfloor < \lfloor n/h \rfloor) + (j < n)$. To maximize f, let $\lfloor j/h \rfloor = \lfloor n/h \rfloor - 1$ and $j \bmod h = h - 1$; except when $n \bmod h = h - 1$ let $j = n$. We can also express this as $j = \lfloor (n+1)/h \rfloor h - 1$.

(c) $h = 9 \Rightarrow f(h, j, 80) \leq f(9, 80, 80) = 51$. (Quick was close.)

(d) In general the worst case running time is $3\lfloor n/h \rfloor + 3h - (n \bmod h \neq h - 1)$. To minimize $g(h) = \lfloor n/h \rfloor + h$, note that when $h \leq \sqrt{n}$ we have $g(h) \leq g(h-1)$, since $n/(h-1) > n/h + 1$; and when $h \geq \sqrt{n}$ we have $g(h) \leq g(h+1)$, since $n/(h+1) > n/h - 1$. Thus the minimum occurs at $\lfloor \sqrt{n} \rfloor$ or $\lceil \sqrt{n} \rceil$. The worst case running time is therefore $6\sqrt{n} + O(1)$.

2. (a) The complete binary tree C_4 with 16 leaves. (C_0 has one node, C_{n+1} has left and right subtree equal to C_n.)

(b) If x is a leaf, $h(x) = 2d(x) = 0$. Otherwise by induction on $h(x)$ we have $h(x) = 1 + \max(h(l(x)), h(r(x))) \leq 2 + \min(h(l(x)), h(r(x))) \leq 2 + \min(2d(l(x)), 2d(r(x))) = 2d(x)$.

(c) T_7 where T_n is defined recursively as follows: T_0 is a single node, T_{n+1} has left subtree T_n and right subtree $C_{\lfloor (n-1)/2 \rfloor}$.

(d) If the tree has depth d then it must contain C_d, so we have $n \geq 2^{d+1} - 1$, where n is the number of nodes. Thus $h \leq 2d \leq 2(\lg(n+1) - 1)$, so h is $O(\log n)$. (Note we only needed the fact that $h(x) \leq 2d(x)$ at the root node.)

3. The algorithm given below is a form of "dynamic programming".

The idea will be to have one array $M[0 : c]$ such that $M[j]$ is 0 if no packing of size j is possible, otherwise $M[j] = k$ means there is at least one packing of size j that contains the kth object but not objects $k+1, \ldots,$ n. Furthermore if $M[j] = k$ then we have $0 \neq M[j - x_k] < k$, i.e. there is a packing of size $j - x_k$ containing only objects from $1, \ldots, k-1$.

A1. [Initialize.] Set $M[j] \leftarrow 0$ for $1 \leq j \leq c$, and set $M[0] \leftarrow -1$ (this is a slight kludge to indicate that we can achieve the empty packing). Set $m \leftarrow 0$.

A2. [Done constructing M?] (At this point the array M has been set up as specified above, but using only the first m objects.) If $m = n$, go to step A4.

A3. [Add new object.] Increase m by 1. Then for $c \geq j \geq x_m$ (in decreasing order of j), *if* $M[j] = 0$ and $M[j - x_m] \neq 0$, set $M[j] \leftarrow m$. Return to step A2.

A4. [Output the result.] Find the largest k in the range $0 \leq k < c$ such that $M[k] \neq 0$. Output this k as the size of an optimal packing. Then repeatedly output $M[k]$ and decrease k by $x_{M[k]}$, zero or more times, until $k = 0$.

The algorithm is "good" because although the packing problem is NP-complete, our algorithm solves it using $O(nc)$ time and O(c) storage. (Why have we not proved that P = NP?)

Artificial Intelligence

1. Satisfactory answers should relate discussion to some specific work in the following areas:

Vision (shadows, 3d, motion)

Language (understanding, generation, dialogue)

Signal understanding (multiple knowledge sources, changes over time, expectations, bottom-up vs. top-down processing)

Inference/problem solving (search for explanations, blackboard model, syntax of possible plays, semantics of plausible plays, common sense reasoning, reasoning about intentions and beliefs, inexact reasoning, reasoning with incomplete information, generalization, analogical reasoning, pattern matching, planning to provide expectations, focus of attention, distributed problem solving, ill-structured problems)

(a) Important aspects of the sport:

The number of players, number of rules, complexity of interactions among players, speed of play, duration of play between pauses, amount of interpretation of actions the commentator must perform, richness of past history of the sport, availability of compiled statistics *on* teams, players, situations — these all make the problem more or less difficult.

It is important to recognize that commenting involves understanding the action and relating it to broader contexts, not just reporting what one sees. Stereo vision and analysis of motion must be recognized as limiting factors in almost any sport except snail racing, and a couple of references to vision research would be appropriate here. Pattern matching is obviously complex. The problem also requires setting up expectations of future actions, using models of events (and objects) to aid the interpretation, reasoning by analogy, and generalizing from past observations.

(b) Information:

(1) Static information.

rules — it would appear essential to provide the program with complete knowledge of the rules for legal play of the game. From the commentator's point of view, these rules will be accessed in appropriate situations; thus they should

be indexed according to the situations in which they are potentially relevant; this suggests that representing them as production rules would be appropriate. Acquiring this type of knowledge will in most cases be quite straightforward — at least, until the time comes to actually code the low-level primitives out of which these are built (e.g. how to test whether the holding was "intentional" or not.)

goals/purposes — The program must be able to relate situations and/or actions to the player's goal of winning and subgoals for achieving that. Thus the program must model the players' problem-solving abilities. This can be arbitrarily sophisticated, or as simple as assuming that they all have a particular weak method driving their actions (such as means-ends analysis).

models of player' intentions — Closely tied with the preceding would be modelling the motivations of the players, in an attempt to relate individual actions to subgoals. This will of course be almost impossible to obtain (or verify) dynamically; at best, it can be built up over time for each player, with rules which modify it under certain circumstances.

strategies — The program must be able to recognize actions as instantiations of strategies and recognize appropriate and inappropriate uses of a strategy. Meta-rules have been used to represent strategy rules.

statistical data — complete compilations of past performance of individuals, teams, team units, leagues.

knowledge of standard shapes and configurations of objects -I.e., the program must be able to recognize things in the world, common states and arrangements of players on the field, etc. One might employ a frame-based (schematized) rep resentation for this knowledge,

model of relative importance of objects, usual behavior, intentions; knowledge of how to synthesize comments based on highly variable sets of events.

library of one-liners and stories — This is easy to obtain, and will add quite a bit to the "humanness" of the program's output and to its continuous stream of reportage.

(2) Dynamic information.

snapshot of a scene with stereo information — just to provide basic information to the program on what objects (and players) are close to one another. Necessary for interpreting what is going on at time *t*. Such information would greatly speed up many of the computations the program will have to perform. An alternative would be to spend even a bit more time, and compute vectors of velocity for each moving entity in the scene. This would be represented differently, say using pointers into a frame-structured corpus. While this kind of knowledge would seem to be trivial to obtain, AI research has revealed it to be a painful, difficult task after all.

actions that have taken place in this game, and the context in which they occurred — must have a representation of patterns found in individual scenes.

patterns of actions — generalizations on sets of prior actions. (Could very quickly pass beyond the state of the art of AI, although there is considerable work on induction.)

(c) Flow of control:

for each time frame from (start of event - 30 min) to (end of event + 15 min)
- input raw tv signal for brief time frame
- find individual objects in the scene
- identify each object, using expectations generated in previous time frames plus strong model of allowable positions for individuals
- focus on most interesting parts of scene
- relate this scene to previous scenes to determine differences
- if no differences and pause is expected then generate blather about objects or patterns that were recently changed. Comments can include reading from canned histories and books of statistics. If unable, then say "What do you think of that, Don?"
- determine interesting differences, interesting patterns
- find plausible explanations of these
- generate comments about the differences/patterns and their explanations, expecially noting rule infractions, scores, unexpected events — comment immediately on high interest events.
- interpret purpose of changes and patterns with respect to known strategies and desirable subgoals
- generate comments on these
- get next time frame

Notice that *you* can be more specific, since you selected a specific sport in (a), and supplied a specific set of information sources in (b), whereas we have tried to remain sport-independent.

(d) Pros and cons of knowledge-based approach:

Understanding requires considerable knowledge of the sport. Published rule book alone is insufficient for generating text for comments. Expertise about interesting changes and patterns, plausible strategies, etc. is clearly required for interesting commentary. Only by cleanly separating inferential procedures from the knowledge that is specific to the sport is there any chance of mapping the program into another sport.

Whole problem is too open-ended to allow capturing sufficient expertise in a program.

Expect that this approach would have a longer initial start-up cost than a non-AI one, but would achieve expert-level performance ultimately more quickly, but would run an order of magnitude *or* three slower.

(e) Two-year project:

Many good problems — any must be sufficiently constrained to allow the following:

modestly small data structures,

reasonably complete knowledge base,

avoiding hard problems that have enmeshed good people for years (e.g. you don't really intend to implement those auxiliary boxes labelled "natural language understander", "speech understander", "discoverer of patterns", etc.)

Systems

1. precedence — Advantages: Small tables. Disadvantages: Only parses a small class of grammars and a small class of languages.

LL — Advantages: Small tables, parses a bigger class of languages than precedence. Disadvantages: Only parses a small class of languages and it is difficult to find the (small) LL grammar for a particular LL language.

LR — Advantages: Parses a large class of grammars and a large class of languages. Disadvantages: Large tables and those tables are hard to generate.

LALR — Advantages: Parses a large class *of* grammars and a large class of languages. Disadvantages: The tables, while not as large as LR tables, are hard to construct.

2. One must generate code for producing values before using the values and one must generate code for all *uses* of a value from a particular type of memory cell before generating code that stores a new value into that cell (or a class containing that cell in cases like indexed arrays).

3. Aliasing is the situation of having two names for a particular value simultaneously active. The most common type of alias is an actual parameter that is also addressable as a more globally named value or two formal parameters that actually represent the same actual parameter. It is considered bad because it allows the unwary programmer to produce obscure bugs, it allows the unwary code optimizer to produce obscure bugs, and it requires the wary optimizer to produce cautious but slow code. In Pascal, variant records without tags are also a major source of aliasing. One could eliminate them and put additional requirements on the use of tags in variant records. One could also change 'VAR' parameters (reference parameters) to 'value-result' parameters or one could try to construct compilers that would detect actual aliasing (as opposed to potential aliasing) and issue error messages.

4. The simplest scheme involves providing a simple but complete symbol table to the debugger and giving the debugger complete value access and change capability at any point in the control flow as well as the usual manipulations of the control flow itself. This scheme restricts what the compiler can do in the way of optimizations such as code motion, strength reduction, storage folding, etc. A more complex scheme allow6 the compiler to do many of the just mentioned optimizations but requires that it supply more extensive information in tables to the debugger so that the debugger can find its way back to variables and points in the control flow or report that such values are undefined at some point or report that some points have changed. Still, global optimizations of several important sorts must be severly constrained. At the current time, this is not much of a limitation since global optimizations are relatively rare.

5. Binding time is the time at which a name gets associated with a particular attribute such as a value or type. For example, with call by value parameters, the binding of the value of the parameter to the formal name of the parameter occurs at the time of the corresponding procedure call while a call by name parameter does not get bound to a value until the formal parameter name is actually used. Another example would be types determined at compile time as in Pascal versus types determined at run time as in Simula. Early binding usually has the advantage of efficient implementation and the disadvantage of lack of flexibility while late binding maintains flexibility and often results in more general and more powerful programs but with more significant run time overhead.

6. If one thinks of the assertions to'be verified as another version of the program, then it is easily seen that the aboue proposal is not a proposal to do a verification system in a different way. In overly simplistic terms, one could say that the verification problem is the program equivalence problem.

7. <statement> ::= <matched statement> |<unmatched statement>
<matched statement> ::= <assignment> | if <expression> then <matched statement> else <matched statement>
<unmatched statement> ::= if <expression> then <statement> | if <expression> then <matched statement> else <unmatched statement>

8. By the explicit sending and receiving of messages indicating the need for or the realization of synchronizing conditions. No other operation6 are needed.

9. Activation record retention means the retaining of activation records for instances of a procedure in a non-stack fashion. Thus we can think of all storage, including activation records, as coming from a heap and having no stack storage for activation records at all. This is a convenient way to organize activation records in procedure oriented operating system based on monitors because process forks

can be implemented simply as the creation of a new chain of activation records in the heap and monitors can queue and dequeue these activation record chains (processes) in a nearly obvious and straightforward fashion.

Numerical Analysis

1. (a) The method of successive substitutions has the general form $x^{(k+1)} := \varphi(x^{(k)})$. A necessary condition for convergence to a root α is that $|\varphi'(\alpha)| \leq 1$. Here we have

$$\varphi(x) = \sinh x - 3\,,\ \varphi'(x) = \cosh x \geq 1\ ,$$

so the suggested method cannot converge.

(b) There are an infinite number of ways of rewriting the given equation in the form $x := \varphi(x)$ which will lead to a convergent method. One obvious way is $x := \sinh^{-1}(x + 3)$, for which

$$\varphi'(x) = \frac{1}{\sqrt{1+(x+3)^2}}.$$

This is ≤ 1 for all x, and in the vicinity of the root $\alpha = 2.38534\ldots$ it is about 0.2. A better way to solve this equation would be by Newton's method,

$$x^{(k+1)} := x^{(k)} - \frac{x^{(k)} - \sinh x^{(k)} + 3}{1 - \cosh x^{(k)}}\ ,$$

which might also be considered a form of successive substitution.

2. (a) To find the change in φ when the data is subject to small changes we first compute

$$\frac{\partial(\cos\varphi)}{\partial u_1} = \frac{v_1 - u^T v u_1/(u_1^2+u_2^2)}{\|u\|_2\,\|v\|_2} = \frac{u_2(v_1u_2 - u_1v_2)}{\|u\|_2^{\ 3}\|v\|_2}\,.$$

Now observe that $v_1u_2 - u_1v_2 = \|u\|_2\,\|v\|_2 \sin\varphi$. Then, using the chain rule for differentiation, we get

$$\frac{\partial\varphi}{\partial u_1} = \frac{-1}{\sin\varphi}\,\frac{\partial(\cos\varphi)}{\partial u_1} = \frac{-u_2}{\|u\|_2^{\ 2}}\,.$$

Since the coordinates occur in complete symmetry similar formulas for the other derivatives follow immediately. If we know bounds for the relative perturbations

in the data, $|\Delta u_1| \leq \delta |u_1|$ etc., we get for the change in φ the approximate bound

$$|\Delta\varphi| \leq \left(\frac{|u_1 u_2|}{\|u\|_2{}^2} + \frac{|v_1 v_2|}{\|v\|_2{}^2} \right) 2\delta \leq 2\delta .$$

This shows that the quantity φ is well-conditioned as a function of the coordinates.

(b) If the formula in (a) is used to compute φ numerically, rounding errors will be introduced. If floating-point computation with a relative precision of ϵ is used then we have

$$|fl(u^T v) - u^T v| \leq (|u_1 v_1| + |u_2 v_2|) 2\epsilon \leq \|u\|_2 \|v\|_2 \, 2\epsilon .$$

This leads to a relative error in $\cos\varphi$ bounded by 2ϵ, and

$$|\Delta\varphi| \approx \frac{|\Delta(\cos\varphi)|}{|\sin\varphi|} \leq \frac{2\epsilon}{|\sin\varphi|} .$$

Thus if φ is small, then rounding errors can cause perturbations in φ which are large in absolute value. (It might even happen that the computed $\cos\varphi$ becomes larger than 1!)

(c) The important difference is that here we compute φ from

$$\psi \equiv 2 \arctan r , \quad 0 \leq r \leq 1 .$$

Rounding errors will occur in the computation of α and β, but these are not magnified since if $0 \leq \psi \leq \pi/2$ then

$$\frac{d}{dr}(\arctan r) = \frac{1}{1+r^2} \leq 1 .$$

Note that the algorithm given here applies to vectors of arbitrary dimension. For vectors in the plane there are simpler algorithms which are stable.

3. (a) The elements in the matrix $A^T A$ are

$$(A^T A)_{kj} = \sum_{i=1}^{m} a_{ik} a_{ij} .$$

If $|j-k| \geq w$ it follows from the properties of A that all terms in this sum are zero and therefore

$$(A^T A)_{kj} \neq 0 \quad \Rightarrow \quad |j-k| < w .$$

According to the general definition of band width given, it follows that the band width of A^TA is (at most) $2w - 1$. (The band width of a symmetrix matrix is more often defined as the maximum number of non-zero elements within the upper triangle in any *row*. With this definition the band width of A^TA would be w.)

(b) For any n-vector x we have $x^T(A^TA)x = (Ax)^T(Ax) = \|Ax\|_2^2$. This must be ≥ 0, with equality only if $x = 0$ or if A is singular. If A is positive definite, then the LL^T decomposition can be computed stably by a form of Gaussian elimination without pivoting. Such a decomposition is called a Cholesky decomposition. This means that the band structure of A^TA will not be destroyed by the elimination process.

(c) Begin by setting $A^TA := 0$. As the i-th row of A is made available, compute each of the quantities

$$a_{ij}a_{ik}, \quad j \leq k$$

where j and k are confined to the nonzero band for row i, and add each such product to $(A^TA)_{jk}$ and $(A^TA)_{kj}$. (In practice, only the upper or lower triangular half of A^TA need be maintained, because of symmetry.)

(d) Using the procedure described in (c), we compute A^TA in approximately $mw^2/2$ multiplications. To compute the LL^T decomposition by Gaussian elimination takes approximately $nw^2/2$ multiplications if $w \ll n$, since no pivoting is required as mentioned in (b). In contrast, if A is a dense matrix (or if you treat it as such), these numbers become $mn^2/2$ and $n^3/6$.

Theory of Computation

1. Let p_1 be an arbitrary Algol program, not containing the variable x. Consider the program p_2 = begin integer x, y; p_1; y ← x; end. Clearly, x is used without being assigned a value if and only if the program p_1 terminates.

2. (a) Have productions: $S \to rR_1 \mid lR_3 \mid \epsilon$, $R_1 \to rR_2 \mid lS$, $R_2 \to rR_3 \mid lR_1$, $R_3 \to rS \mid lR_2$. Note this is a regular grammar.

(b) The language contains all strings over $\{n, s, e, w\}$ such that the number of n's is equal to the number to s's and the number of w's equal to that of e's. The nonempty strings are generated by the context-sensitive grammar with start symbol Z and productions $Z \to$ NS | EW | NSZ | EWZ, XY → YX for all X and Y in $\{N, S, E, W\}$, and $E \to e$, $N \to n$, $S \to s$, $W \to w$.

(c) We may assume without loss of generality that the path begins and ends going north. Replace the last four productions of (b) by $NE \to r\overline{E}$, $NW \to l\overline{W}$, $\overline{N}E \to r\overline{E} \mid rl$, $Nw \to l\overline{W} \mid lr$, $\overline{E}S \to r\overline{S}$, $\overline{E}N \to l\overline{N}$, $\overline{S}W \to r\overline{W} \mid rr$, $\overline{S}e \to l\overline{E} \mid ll$, $\overline{w}N \to r\overline{N}$, $\overline{W}S \to l\overline{S}$.

Note: The languages in (b) and (c) are not context-free, which can be proved by considering the intersection of the language with an appropriate regular expression.

However, this is beyond the scope of the problem.

3. For fixed n we can define $n + 1$ functions a_i, $0 \leq i \leq n$ as follows:

$$a_0(y) = y + 1 \quad \text{(the successor function)};$$
$$a_{i+1}(0) = a_i 1) = 0 + 1 + \cdots + 1 \text{ (repeated } a_i(1) \text{ times)};$$
$$a_{i+1}(y + 1) = a_i[U_2^2(y, a_{i+1}(y))).$$

We can show by induction that $a_i(y) = a(i, y)$; furthermore all a_i are primitive recursive.

4. We can reduce the directed Hamiltonian circuit problem to the stated problem as follows. Given a directed graph, consider the vertices as the states of a finite automaton and the (directed) edges as transitions between the states. Associate with each vertex a distinct terminal symbol, and label each edge entering that vertex with this symbol. Choose any vertex as the start vertex, and make that start vertex the only final state,

The resulting finite automaton accepts a string that contains each terminal symbol exactly once if and only if the original graph has a directed Hamiltonian circuit. Because of the correspondence between finite automata and regular grammars, we have in fact shown that the stated problem is NP-hard even for regular grammars. Moreover, it is in NP (and thus NP-complete) since we can nondeterministically check all $n!$ strings to see if they are in the language.

5. (a) In the construction of the sequence $q'_1 q'_2 \ldots q'_n$ for $j < i \leq m$, consider the case $p_i = r_{i-1}$. Since $p_i \notin S_{i-1}$, we save a page fault here.

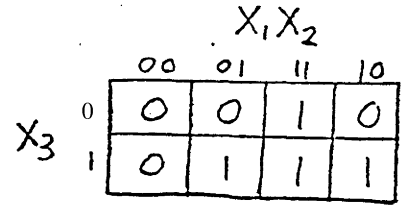
We lose a page fault at time $m + 1$ iff $p_{m+1} = q$, but we always gain at least one page fault the first time $q_j = p_k$ since $r_{k-1} = q_j$. If $p_{m+1} = q$ we have $m + 1 > k$, so there is never a net loss.

(b) The periodic page trace $a_1 a_2 \ldots a_{b+1} a_1 a_2 \ldots a_{b+1} a_1 a_2 \ldots$ causes page faults every time with LRU, but only every bth time with our optimal strategy (after the first period). Incidentally, such worst-case behavior is *not* so uncommon; it occurs during long iterations.

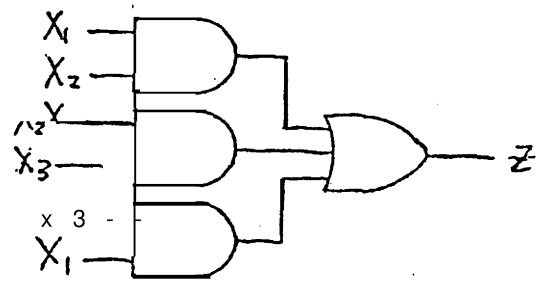
Hardware

1. (a) The truth combinations and Karnaugh map are shown below.

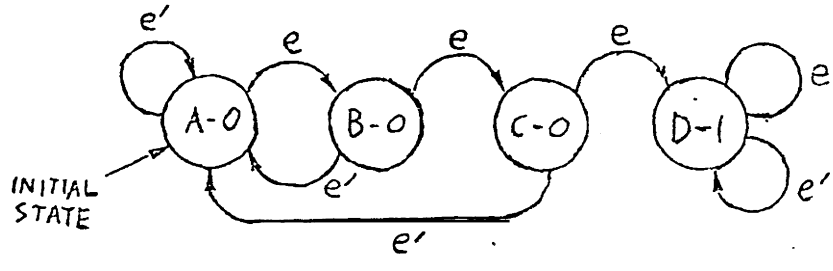
X_1	X_2	x_3	Z
0	0	0	0
0	0	1	0
0	1	1	1
0	1	0	0
1	1	0	1
1	1	1	1
1	0	1	1
1	0	0	0



A logic expression for this circuit is $z = x_1x_2 + x_2x_3 + x_3x_1$. A circuit for V is shown below.



(b) A state diagram for the desired circuit is



where e = $x_1 \oplus z$ ($\oplus$ denotes exclusive-or). This circuit will be designed for clocked (synchronous) pulse-mode operation. Thus, it is assumed that each clock pulse is "long enough to cause the appropriate flip-flops to change state". It is also assumed that each clock pulse is "short enough so that it is no longer present at the circuits which generate the flip-flop input signals when the change in flip-flop outputs has propagated to the input circuitry" (from p. 205 of *Introduction to the Theory of Switching* Circuits by E. J. McCluskey). Furthermore, it is assumed that the signals x_1, x_2, x_3, z, and e do not change while the clock is active (c = 1).

The combined state and output table is shown on the left below, where T is the level-output of the sequential circuit. Since there are four states, at least 2 internal variables are required. Let the state assignments be A $= y_1'y_2'$, $B = y_1'y_2$, $C = y_1y_2$, and $D = y_1y_2'$. Note that $T = y_1y_2'$, since the output is 1 only when the circuit is in state D.

The transition table is shown on the right below,

S	c=0	e=0	e=1
initial state → A	(A), 0	(A), 0	B, 0
B	(B), 0	A, 0	C, 0
C	(C), 0	A, 0	D, 0
D	(D), 1	(D), 1	(D), 1

S; T

Y_1Y_2	c=0	e=0	e=1	T
A - 00	00	00	01	0
B - 01	01	00	11	0
C - 11	11	00	10	0
D - 10	10	10	10	1

Y_1Y_2 T

Using set-reset flip-flops, the encoded excitation table is

Y_1Y_2	c=0	e=0	e=1
00	r, r	r, r	r, S
01	r, s	r, R	S, s
11	s, s	R, R	s, R
10	s, r	s, r	s, r

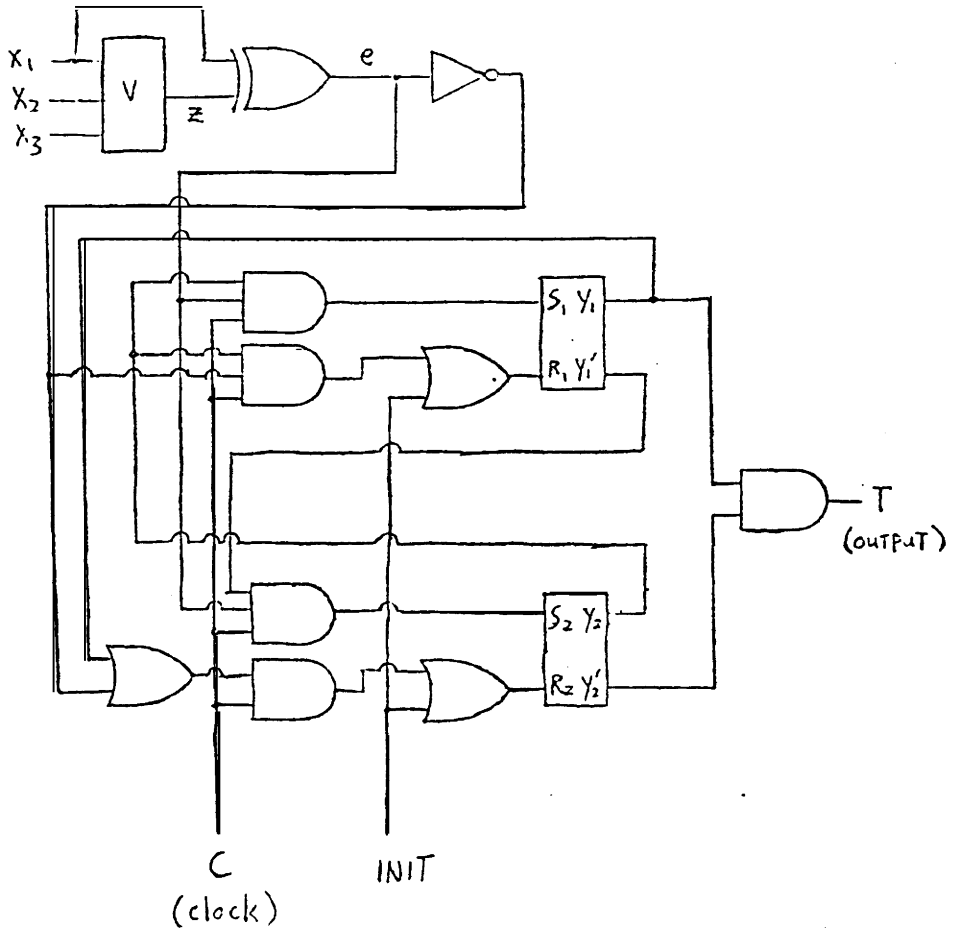
S_1R_1, S_2R_2

By inspection of the excitation table,

$$S_1 = cey_2, R_1 = cc'y_2, S_2 = cey_1', \text{ and } R_2 = c(e' + y_1).$$

To implement the initial condition, the reset inputs are changed by ORing in an initialization pulse INIT. Thus $R_1 = cc'y_2 +$ NT, and $R_2 = c(e' + y_1) +$ INIT. The INIT pulse is assumed to be applied before circuit use begins.

The entire circuit is shown at the top of the next page.



2. (a) When adding two 2's complement numbers, overflow occurs if and only if the signs of the inputs are the same, and the sign of the output is not equal to the sign of the inputs. Since the sign of a 2's complement number is the most significant bit (MSB), we have

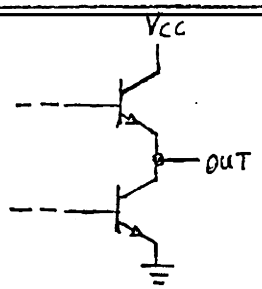
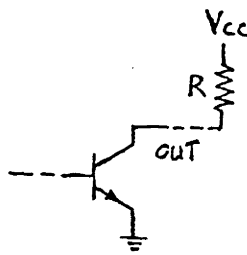
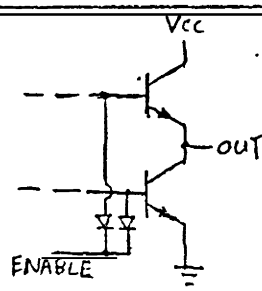
$$\text{overflow} = \text{MSB}(A)\text{MSB}(B)\text{MSB}(S)' + \text{MSB}(A)'\text{MSB}(B)'\text{MSB}(S)$$

where S is the sum of A and B.

Overflow can also be detected as the exclusive-or of carry-in with carry-out at the most significant bit of the adder circuit.

(b)
$$\begin{array}{r} 01111111 \\ +00000001 \\ \hline 10000000 \end{array}$$

3.

TTL OUTPUT	SIMPLIFIED CIRCUIT	ADVANTAGES	DISADVANTAGES
Standard		simple	cannot be used for data busses
Open Collector		can be used for data busses; "wired-and" logic function at outputs; also useful for analog interfacing.	external pullup resistor required; passive pullup results in longer risetime and poor fan-out capability.
Three - State (1, 0, hi-impedance)		well suited for data busses because of active pullup and third (high-impedance) state	extra input pin needed for enable function; control failure at enables can result in ambiguous output if two outputs "fight".

SPRING 1979 PROGRAMMING PROJECT: CODE GENERATION AND OPTIMIZATION

Due: 12:00 p.m., Tuesday, April 10, 1979
Polya 254

Outline:

Let M be a simple computer that performs arithmetic operations on integers and has a memory addressed by positive integers. M can perform only register-register arithmetic operations, and has no jump instructions, indexing or indirect addressing. The goal is to write a simple code generator named COP which generates efficient code for M .

Input to COP:

The input to COP consists of triples. Each triple consists of an operator and two arguments. Triples are numbered consecutively from 1 ; the n-th triple is said to have triple number Tn . The result of a triple is a number, which may be referred to in subsequent Triples by using its triple number as an argument.

Here are the operators available in triples:

Operator	Result
+	arg_1 + arg_2
-	arg_1 - arg_2
*	arg_1 * arg_2
/	arg_1 / arg_2 (integer division)
=	arg_1 , with side effect of arg_2 := arg_1 .

Each argument, arg_1 or arg_2 may be an integer constant, a memory address An (with n in the range 0 - 99), or the triple number Tn of a previous triple.

For example, the following triples have the same effect as the statement X := X*Y + X*Y + 3 ; given that X and Y are allocated memory locations 0 and 1:

	Operator	arg_1	arg_2
1	*	A0	A1
2	*	A0	A1
3	+	T1	T2
4	+	T3	3
5	=	T4	A0

Output of COP:

The output of COP is a sequence of instructions for M . M has four general registers R0 , R1 , R2 , R3 and two hundred memory locations (for data) numbered 0 . . . 199 . Below r and s denote registers, m a memory address. Read C(x) as "the contents of x ". M has the following instructions:

LOAD	r,m	load register r with C(m)
LOAD1	r,k	load register r with the value k
STORE	r,m	store C(r) into memory location m
ADD	r, s	put C(r)+C(s) into register r
SUB	r, s	put C(r)-C(s) into register r
MULT	r, s	put C(r)%(s) into register r
DIV	r, s	put C(r)/C(s) into register r

For example, the triples could be optimized into:

INSTRUCTION		COMMENT
LOAD	R0,0	triple 1
LOAD	R1,1	triple 1
MULT	R0,R1	triple 1
ADD	R0,R0	triple 3
LOAD1	R1,3	triple 4
ADD	R0,R1	triple 4
STORE	R0,0	triple 5

Objective:

Assume that ADD, SUB, MULT, DIV AND LOAD1 each take one unit of time to execute and that LOAD and STORE each take two units. Your optimizer should try to minimize the time required to run the output code, but the optimizer should be efficient enough to be practical in a compiler and should always generate correct code.

Method:

Two techniques you can try are:

(1) Common subexpression elimination
(e.g. (X*Y+Z)+(X*Y+Z) should lead to only one computation of (X*Y+Z).

(2) Constant folding
(e.g. X+1+2 should be converted at compile time to X+3).

Try to make efficient use of M's registers. Your code generator should be prepared for expressions complex enough to require storing temporary results in memory, using addresses in the range 100 ... 199 (remember, 0 . . . 99 are reserved for variables).

Possible algorithms are sketched in Aho and Ullman, Principles of Compiler Design (1977), sections 12.3-12.4 and 15.4-15.6, and Gries, Compiler Construction for Digital Computers (1971), sections 17.2 and 18.1. Both books are on reserve in the Math. Sciences Library. However a fully satisfactory program can be written without referring to books at all.

Do not try to use unusually sophisticated optimizations (e.g., avoid sections 12.5 and 15.7 of Aho and Ullman). Attempt only what you know you can finish on time. To get a feel for the problem, you might begin by writing a code generator that performs no optimization. If you can't get your optimizer to work, but do get a code generator working, then turn it in.

Documentation.

It is not sufficient that your program work; the graders must be able to see that it works. So you must document your program and its output. Explain which algorithms you used and give references; outline the general layout of the program; describe how to interpret its output. In the program itself, use descriptive variable names; use proper indentation; insert comments where appropriate.

To make the generated code readable, it should include such comments as "store in temporary variable" or "triple n" (i.e., COP should put these comments in its output). Also COP should print the number of time units required to execute the code it generates. If your optimizer produces an optimized set of triples (rather than going directly to assembly code), it should print these.

Test data for your program will be given out in two sets. The first set is attached here. The second set will be available after 9:00 a.m. on Monday, 9 April, in Polya 254. Turn in your source program, associated documentation and its output from all the test data,

Questions:

If you have questions about the problem (besides how to solve it!), contact

Larry Paulson	LP@ SUAI	497-4971	327-1104

or

Lloyd Trefethen	LNT@ SUAI	497-4368	325-5396

1	*	3	A2
2	/	5	A3
3	+	Al	4
4	+	T1	T3
5		T4	T2
6	*	T5	3
7	+	6	T6
8	-	T7	4
9	=	T8	Al
10	*	4	Al
11	/	A2	A3
12	+	T10	T11
13	*	4	Al
14	+	T12	T13
15	/	T14	2
16	+	2	Al
17	-	T15	T16
18	=	T17	A2
19	*	A4	A5
20	-	A3	A2
21	/	Al	A2
22	/	T20	T21
23		T19	T22
24	*	A8	A9
25	*	A10	Al1
26	-	T24	T25
27	*	T23	T26
28	=	T27	A3

1	+	A1	A2
2	*	3	A2
3		A3	A4
4	/	5	A3
5	+	A5	A6
6	+	A1	4
7		A7	A8
8	+	T2	T6
9		A9	A10
10		T8	T4
11	+	A11	A12
12	*	T10	3
13		A13	A14
14	+	6	T12
15	+	A15	A16
16		T14	4
17	*	T1	T3
18	=	T16	A1
19	/	T5	T7
20	*	4	A1
21	/	T9	T11
22	/	A2	A3
23	*	T13	T15
24	+	T20	T22
25		T17	T19
26	*	4	A1
27	+	T21	T23
28	+	T24	T26
29	/	T25	T27
30	/	T28	2
31	-	A13	A14
32	+	2	A1
33	+	A15	A16
34	-	T30	T32
35	*	T31	T33
36	=	T34	A2
37	=	T29	A4
38	*	A4	A5
39	+	A1	A2
40	-	A3	A2
41	-	A3	A4
42	/	A1	A2
43	+	A5	A6
44	/	T40	T42
45	-	A7	A8
46	-	T38	T44
47	*	T39	T41
48	*	A8	A9
49	/	T43	T45
50	*	A10	A11
51	-	T47	T49
52	-	T48	T50
53	/	T35	T51
54	*	T46	T52
55	=	T53	A1
56	=	T54	A3
57	*	3	4
58	/	8	2
59	/	T57	T58
60	=	T59	A5

REMARKS ON THE GRADING OF THE PROGRAMMING PROJECT

The programs we received varied greatly in sophistication and in the methods of optimization employed. Most people succeeded in generating correct code for "COP" and performed at least one kind of optimization, however.

Grading came down to three broad areas:

(1) Code Generating and Optimizing Algorithms: Many people implemented DAGs (directed acyclic graphs) or a related technique; a few more did not. The majority performed some sort of optimizing on the triples themselves before generating code, often including a reordering of the triples. Reordering could have a dramatic effect on the efficiency of code generated for the second assigned input set, but we deemphasized this somewhat and weighed also less dramatic forms of optimization, such as constant folding and common subexpression elimination.

Typical "scores" for masters passes were 70/230 COP time units for the first/second input set, and for Ph.D. passes 62/185 time units. However, these numbers in general counted less than you may have expected in determining your performance.

(2) Programming Style and Efficiency: The following inelegancies appeared too often:

- Use of obscure numbers like "4" where a macro like "MULTIPLY" would have been much clearer.
- Unnecessary string operations.
- Duplication or near-duplication of code.
- Lack of division of major tasks into smaller procedures.
- Lack of comments in code.
- Inefficient use of arrays where linked structures would have been more appropriate, and/or unnecessary linear searches where hashing or a better data representation could have saved time. It's all right to cut corners on a thing like hashing, but if you do, you should mention in your documentation that there is a better alternative.

Some of these complaints may be controversial. Clarity, however, is a must, and too many of the programs were hard to read.

(3) Documentation: This was the area most often disappointing. Providing good documentation is absolutely essential to doing well on the programming project. This means:

- Describe your program fairly completely. (Say, at least four pages.)
- Isolate your central data structures and the flow of your program so that the readers don't have to figure them out.
- Documentation should be organized, not just five pages of text.

- Mention more efficient alternatives that you have chosen not to implement. If you think that constant folding is useless in a real compiler, then you must say so, so that we know why you didn't implement it.

In general, too many people seemed to have left tidying up to the program and documenting it to an hour or two at the end. An additional two hours on such "cosmetics" would have given several. such people a higher grade. We are not just being fussy here -- the programming project aims to show that you can produce efficient programs in an environment where they have to be understood and modified by other people.

SAMPLE PROGRAM:

As a sample solution we have duplicated one of the best programs that was turned in, that of Magic Number 4502. This person did use DAGs, and got excellent results: 55 and 133 COP time units, respectively, for the two input data sets.

Here are some strengths of this program;

- Excellent optimization techniques, including common subexpression elimination, reordering of triples, good allocation of registers based on next use information, and detection of commutativity.
- Clear programming style in most respects, including use of macros for integer parameters, extensive commenting within the code, and clear structuring of each procedure.
- Fairly clear documentation up front.

Here are some weaknesses:

- A small bug in the treatment of assignment statements made the first line of code for input set 2 appear before other references to A5 -- an error. #4502 noticed this error at the last minute and commented on it.
- String variables are used more than necessary.
- The program is put together as a sequence of blocks rather than as a collection of procedures called by a small segment of main program. This makes the program less readable globally than it is locally.

The sample program will not be available for several days, as it is being reproduced at SEL Publications.

Winter 1979/80 COMPREHENSIVE EXAMINATION

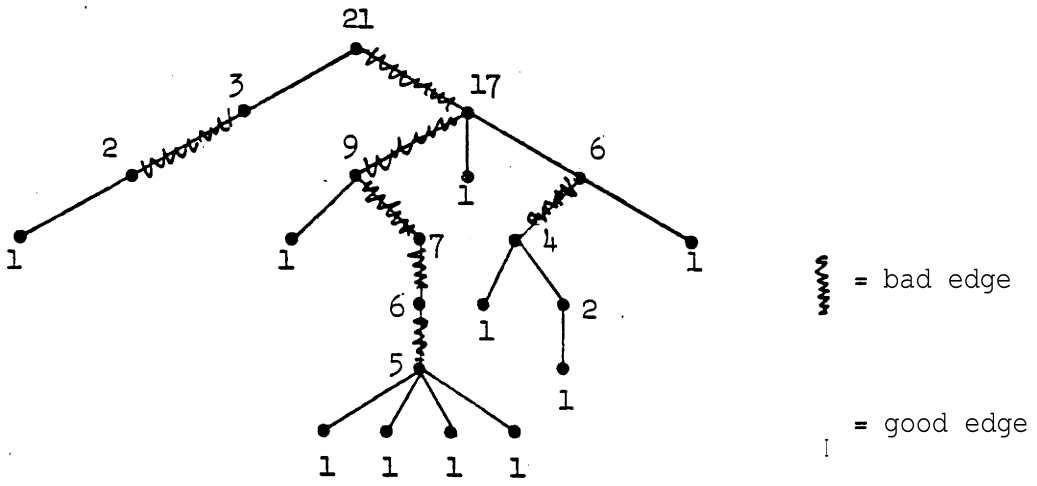
ALGORITHMS AND DATA STRUCTURES

Problem 1. (10 points).

Write an algorithm in pidgin algol which takes as inputs two sorted arrays $x_1 \leq x_2 \leq \ldots \leq x_n$ and $y_1 \leq y_2 \ldots y_n$ andanumber s. The output will be a pair (i,j) such that $x_i + y_j =$ or a statement that no such pairs exists. Your algorithm should run in O(n) time.

Problem 2. (20 points).

Consider a rooted tree. Define the weight of a node x to be the number of nodes in the subtree rooted at x . Let n be the weight of the root. (Note: n is the total number of nodes in the tree.) Consider an edge between a node v and a child of that node, w . Call the edge "good" if 2 x (weight of w) $\leq$ weight of v and "bad" otherwise.



(1) (5 points). Prove that at most one of the edges from a node v to its children is bad.

(2) (5 points). Prove that the path from any leaf to the root contains at most $\log_2 n$ good edges.

(3) (10 points). The least common ancestor of a pair of nodes x,y is defined to be the node of greatest distance from the root which is on both the path from x to the root and the path from y to the root.

Find an algorithm which takes as input two nodes x and y and finds their least common ancestor in time $O(\log n)$. Describe your algorithm in English. In order to do this, your algorithm will have to have some representation of the tree and/or some pre-computed information about the tree. These two together are called the data structure for the tree. Specify the exact data structure your algorithm assumes has been given it about the tree. The data structure should only use $O(n)$ words of memory.

Hint: (1) above implies that a set of connected bad edges must be a linear sequence. A sequence of bad edges is called a bad path. Let bad paths and good edges play a role in your data structure and algorithm.

Problem 3. Sorting by Flipping. (15 points),

Given a sequence that is a permutation of the numbers 1 through n, a flip consists of selecting a set of contiguous elements at the left end of the sequence and reversing their order.

Example: Given 3 4 6 9 8 2 1 7 5, we can flip the first four elements to get

9 6 4 3 8 2 1 7 5.

(a) (5 points). Prove that any permutation of length n can be arranged into sorted order in at most $2n-2$ flips.

(b) (10 points). Prove that, for every n, at least n-1 flips are necessary to sort in the worst case.

Problem 4. (15 points).

An undirected graph is called marked if every edge has either a + or a - sign. A marked graph is balanced if the product of signs around every cycle is positive (i.e., there are an even number of - signs on every cycle). Give a linear-time test for balance. Your algorithm can be specified in English but you must give an argument why the program is correct and why it requires only linear time.

ARTIFICIAL INTELLIGENCE

Problem 1. (10 points).

(a) What are the three classes of theorem in the PLANNER language? Describe them briefly, indicating how they are invoked.

(b) To which class or classes is MYCIN's approach most similar?

(c) Same question but with respect to HEARSAY-II.

Problem 2. (10 points).

The following constants, function symbols, and relations can be used for encoding facts about chess as statements in predicate calculus. They will be used for both this problem and Problem 3.

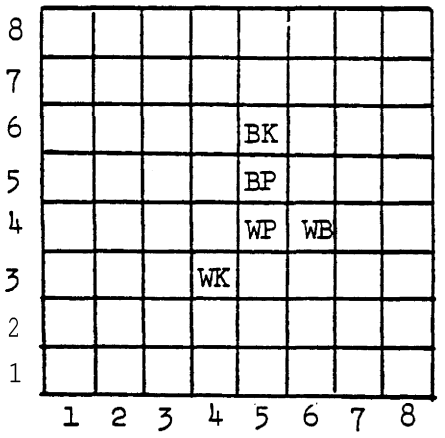
EMPTY(square) -- there is no piece on the square
ON(piece, square) -- the indicated piece is on the indicated square
WHITE(piece) -- the piece is white
BLACK(piece) -- the piece is black
ROW(square, integer) -- the square is in the row indicated
COL(square, integer) -- the square is in the column indicated

ISA(object, set) -- the object is a member of the indicated set
SUBSET(set1, set2) -- set1 is a subset of set2
$<$, $\leq$, $>$, $\geq$, = , + , - , 1, 2, 3, ... -- their usual interpretations

SQUARE, PIECE, PAWN, ROOK, KNIGHT, BISHOP, QUEEN, KING,
WK, WQ, WB-1, WB-2, WN-1, WN-2, WR-1, WR-3, WP-1, ...
BK, BQ, BB-1, BB-2, BN-1, BN-2, BR-1, BR-2, BP-1, ...

all with the obvious interpretations.

using this vocabulary, encode the location of the black king in the following board position as a formula in the predicate calculus.



(b) Write down the predicate calculus statement asserting that only one piece may be on a square at a time and that a piece may be on only one square.

(c) Write down the conditions under which a white P may be promoted to queen.

You may ignore the possibility of king in check.

Problem 3. (10 points).

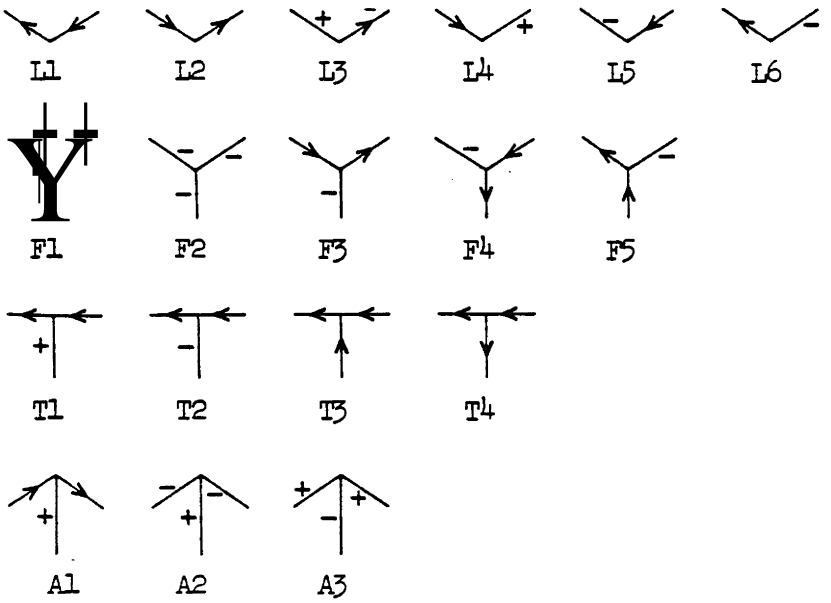
In his thesis and textbook, Winston describes a program able to learn concepts from sequences of examples and near misses.

(a) Is it possible to teach Winston's program the rule about queening pawns? If so, present a training sequence and indicate what each example or near miss teaches and what assumptions are necessary. If not, why not?

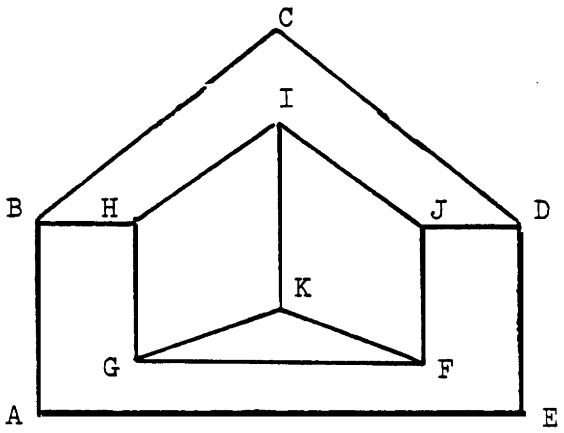
(b) Same question for castling.

Problem 4. (10 points).

The following is the **Huffman-Clowes** label set for labeling **line** drawings (taken from Winston).



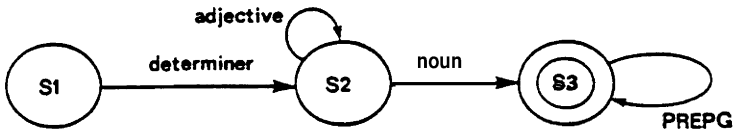
(a) Using this label set, produce a labeling of the following drawing.



(b) Suppose **Waltz's** constraint propagation algorithm were used on the example (still with the **Huffman-Clowes** label set) and that new nodes were considered in the alphabetic order shown. What would be the label sets for each of the nodes Mediately before H is considered? How about immediately before I ? (You may assume that segments AB , BC , CD , DE , and EA are known to be boundaries.)

Problem 5. (10 points).

The following recursive transition network for noun groups is taken from Winston's book.



Using the chess domain, produce an example of each of the following, (you need not restrict yourself to the vocabulary of Problems 2 and 3.)

(a) a noun phrase that exercises all the arcs in the network.

(b) a syntactically legal phrase not accepted by the network.

(c) a syntactically illegal phrase accepted by the network.

(d) a syntactically unambiguous noun phrase that is semantically ambiguous in the-board position of Problem 2.

(e) a syntactically ambiguous noun phrase that is semantically unambiguous in the board position of Problem 2,

Problem 6. (10 points).

(a) What's the difference between the branch and bound search method and the A* algorithm?

(b) Which of the following methods would be most suitable for trying to find the combination to a safe: branch and bound, hillclimbing, depth-first search?

(c) Is an alpha-beta search in all cases more efficient than a full minimax search? If so, why? If not, show a counterexample.

(d) Suppose a one-armed robot were trying to adjust the contrast and brightness controls on his TV. Using only hillclimbing, is he guaranteed to find the optimal picture? Explain very briefly.

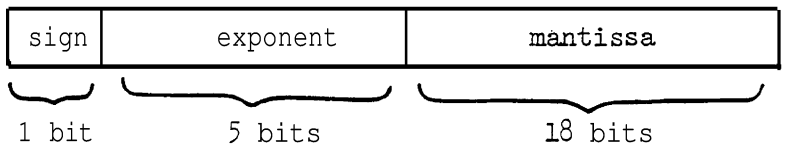
Problem 7. (0 points).

What are two major goals of AI research?

HARDWARE

Problem 1. (15 points).

Suppose we want a component to be attached to a CPU which will be used to compute reciprocals. The floating point numbers used by our CPU are normalized with an octal base. That is, the format looks like:



where the mantissa has an octal point to its left, and the value represented is

$$\text{sign} \times 8^{\text{exponent}} \times \text{mantissa} .$$

(a) (3 points). What range of numbers can be represented if the exponent is 0 and the sign is positive?

The reciprocal machine takes as input some number (to be determined later) of the leading bits of the mantissa. It uses these bits as an address into a ROM. The word stored at that location in the ROM is a fixed point binary number which is within 2^{-7} of the reciprocal of the input, i.e., it has 6 bits to the right of the binary point.

(b) (8 points). What is the minimum number of bits of the mantissa that must be used so that the output of the ROM is within 2^{-6} of the exact reciprocal of the original mantissa?

(c) (4 points). How many words are there in the table?

Problem 2. Sequential Circuit Design. (20 points).

Design a 5 bit serial to parallel converter which includes a generated parity bit output and a data available output. The parity bit is to be a one if there are an odd number of ones in the received 5 bits and a zero if there are an even number of ones. The data available line should be a one if the parallel output lines hold valid data, otherwise it should be a zero.

The inputs to your circuit will be a serial input line, a clock line which has a clock pulse in the middle of each data bit in the serial input line, and a reset line which sets up the circuit so that the next 5 bits received are the ones that will be converted to parallel.

The outputs are the 5 parallel output lines, one line for the generated parity bit, and one line for the data available signal.

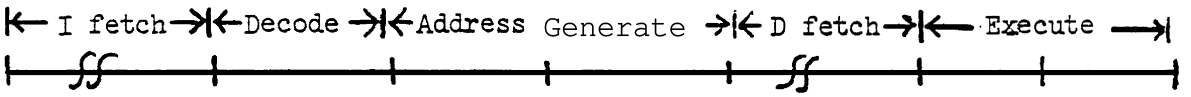
The circuit should use AND, OR, NOT, and XOR gates as well as" D flip flops which have the set, reset, clock, and D lines as inputs and the Q and $\bar{Q}$ outputs available. The set, reset, and clock lines are all negative edge triggered.

The circuit should be able to run continuously, i.e., do not assume a manual reset before each 5 bits.

Problem 3.(13 points).

Suppose a CPU has 4 memory modules which are interleaved together in order to provide fast access for the CPU. Assume that the CPU decodes one instruction per internal cycle and that the following information is true about it:

(1) internal cycle = 100ns
(2) 0.62 instruction memory fetches/instruction
(3) 0.78 data memory fetches/instruction
(4) singe instruction interpretation. Note; each interval in the following time line represents one internal processor cycle. The time interval for the memory fetches is given below.



(5) Combined interleaved memory has an access time =580 ns; cycle time = 100 ns. Note : the CPU is pipelined and creates memory requests by anticipation, therefore it does not wait for memory to respond before issuing the next request. If the CPU were to wait for memory to respond to each request it would take 580 ns, but in pipelined operation words come in every 100 ns from memory.

(a) (3 points). The request rate to memory (maximum) is _____ MAPS. (Million Accesses per Second.

(b) (10 points). The peak performance of the CPU is _____ MIPS. (Million Instructions per Second.) If BC (Branch on Condition) occurs with frequency 0.32, and no "go-ahead-on-branch" strategy is used by the address anticipation mechanism, the resultant CPU performance will be _____ MIPS.

(Please show all work.)

Problem 5. (12 points).

(a) (3 points). Explain the operation of a cache briefly.

(b) (3 points). Explain the advantages and disadvantages between a hard-wired CPU and a micro-coded CPU.

(c) (6 points). What are some advantages and disadvantages of the following logic families.

(1) (2 points). TTL

(2) (2 points). CMOS

(3) (2 points). NMOS

Problem 1. (15 points).

One of the quantities which appears in the error bounds for Gaussian Elimination is the growth factor. For factoring an $n \times n$ matrix $A = \{a_{ij}\}_{i,j=1,n}$, this factor is defined as

$$G = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}$$

where $a_{ij}^{(k)}$ is the element in the (i,j)-th location at the k-th step of the elimination process.

Consider the problem of factoring a tridiagonal matrix of the form

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}$$

Show that the following results hold:

(a) If partial pivoting is not used, G can be aribtrarily large.

(b) For the special case when A is also diagonally dominant, $G \leq 2$, independent of n, even if partial pivoting is not used.

Problem 2. (15 points).

Consider the following two algorithms for computing $\sum_{j=1}^{N} x_j$:

(1) The usual summation algorithm:

```
s := 0
for j = 1,2,...,N do
    s := s+x_j .
```

(2) The pairwise summation algorithm:

Define $S(i,i) := x_i$ for $i = 1,2,\ldots,N$

$$S(i,j) := S\left(i, \left\lfloor \frac{i+1}{2} \right\rfloor\right) + S\left(\left\lfloor \frac{i+1}{2} \right\rfloor + 1, j\right) \quad \text{for } i < j .$$

Compute $S(1,N) = \sum_{j=1}^{N} x_j$ by applying the above definitions. For example, with $N = 8$ the pairwise algorithm computes

$$[(x_1 + x_2) + (x_3 + x_4)] + [(x_5 + x_6) + (x_7 + x_8)] .$$

Assume that the computer arithmetic is such that

$$fl(x_1 + x_2) = x_1(1+\epsilon_1) + x_2(1+\epsilon_2)$$

with

$$|\epsilon_i| < u \quad \text{for } i = 1, 2 .$$

(a) Show that if $S_1 = \sum_{j=1}^{N} x_j$ is computed using the usual summation algorithm, then

$$fl(S_1) = \sum_{j=1}^{N} x_j(1+\beta_j)$$

with

$$|\beta_j| < Nu + O(u^2) .$$

(b) For the special case when N is a power of 2 , show that if $S_2 = \sum_{j=1}^{N} x_j$ is computed using the pairwise algorithm, then

$$fl(S_2) = \sum_{j=1}^{N} x_j(1+\gamma_j)$$

with

$$|\gamma_j| < u \log_2 N + O(u^2) .$$

Problem 3. (15 points).

Suppose that Newton's method is being used to generate a sequence of approximations $x_1, x_2, \ldots$ to a zero x_* of a smooth function f(x) given an initial "guess" x_0 . --

(a) Show that $x_* - x_n = -f(x_n)/f'(\xi_n)$ for $x_* \leq \xi_n \leq x_n$ or $x_n \leq \xi_n \leq x^*$. Using this relation, obtain an estimate of $|x_* - x_n|$ in terms of $|x_{n+1} - x_n|$. Is the quantity $|x_{n+1} - x_n|$ usually a good error estimator to determine the termination of a Newton iteration?

(b) Suppose that we obtain the following behavior of the differences $|x_{n+1} - x_n|$ and that the estimate established in (a) above is valid. Set

$$r_n = |x_{n+1} - x_n| / |x_n - x_{n-1}| .$$

n	$\lvert x_{n+1} - x_n \rvert$	r_n
0	.108	-
1	.075	.692
2	.052	.690
3	.035	.681
4	.024	.675
5	.016	.671
6	.011	.669
7	.007	.667
8	.005	.667

What is the apparent order of convergence? What is the apparent rate of convergence? Recall that if $|x_{n+1} - x_*| / |x_n - x_*|^p < C$, then C is called the rate of convergence (or asymptotic error constant) and p is the order of convergence. Give a plausible explanation of this behavior assuming that f is infinitely differentiable. How might you increase the order and/or rate of convergence while still using Newton iteration?

Problem 4. (15 points).

Let $f: R \to R$ be a uniformly Lipschitz continuous function with Lipschitz constant K (i.e., $|f(x) - f(y)| \leq K|x-y|$ for all x and y). Define $x_n = f(x_{n-1})$ for $n > 0$ with x_0 given as data. Suppose we compute $y_n = f(y_{n-1}) + \epsilon_n$ for $n > 0$ with y_0 given. ϵ_n represents an error introduced at the n-th step and $x_0 - y_0$ represents an intial error in measurement, etc. Show that

$$|x_n - y_n| \leq K^n |x_0 - y_0| + \max_j |\epsilon_j| \left| \frac{K^n - 1}{K-1} \right|$$

if $K \neq 1$.

SYSTEMS

Problem 1. Computer Language Syntax. (10 points).

In some languages (FORTRAN, PL/1) exponentiation (↑ or **) is right associative. A unary minus has lower precedence (binds less tightly) than exponentiation. Other operations { +,-, *J /,), (} bind as expected.

(a) (3 points). Give an example of the effect of right association with an example.

(b) (7 points). Write in BNF the syntax rules for an expression EXPR for all the operators mentioned above beginning with the symbol (identifier) .

Problem 2. Paging. (15 points).

(a) (5 points). Describe the difference between a demand paging algorithm (DPA) that selects the least recently used page in the system for removal and the working set algorithm (WSA).

(b) (5 points). What is needed to make DPA work well in a multi-user system?

(c) (5 points). What is needed to make WSA work well in a multi-user system?

Problem 3. Cooperating Processes. (15 points).

Monitors, as defined by Brinch-Hansen are available in the computing system you are using.

Use such monitors to control message buffering among independently scheduled processes. Sketch the using programs, and write the code of the monitor prototype itself in sufficient detail to allow implementation without ambiguity.

Problem 4. Computer Languages. (10 points).

A criterion in the design of the PASCAL language was the desire to avoid structures that cannot be fully compiled prior to execution.

(a) (5 points). Why is this a desirable goal?

(b) (5 points). What are some of the liabilities of this design?

Problem 5. Language System. (10 points).

PASCAL language systems store NEW records into an area called the heap. List some (3) alternatives for management of the heap and mention succinctly problems to be considered with the listed alternatives.

THEORY OF COMPUTATION

Problem 1. Logic. (12 points).

Consider the following proof system for implicational propositional logic.

AXIOM SCHEMES: $\vdash P \supset P$

$\vdash ((P \supset P) \supset Q) \supset Q$

$\vdash ((P \supset Q) \supset R) \supset (P \supset (Q \supset R))$

INFERENCE RULE:

$$\frac{\vdash P \qquad \vdash P \supset Q}{Q} \qquad \text{(MODUS PONENS)}$$

(a) (4 points). Find a truth function interpretation of " $\supset$ " for which the above axioms and rules are sound, but which differs from the standard truth-function interpretation of "$\supset$".

(b) (4 points). Exhibit a formula with "$\supset$" as its only connective which is valid for the standard interpretation of "$\supset$", but not for the interpretation given as the answer to part (a).

(c) (4 points). Show that the formula given as the answer to part (b) is not provable in the above proof system (thereby showing the incompleteness of the system for implicational propositional logic). Be precise!

Problem 2. Automata and Languages. (12 points).

Is the set of decimal representations of positive integers (read from left to right) divisible

(a) by 2 [2 points]

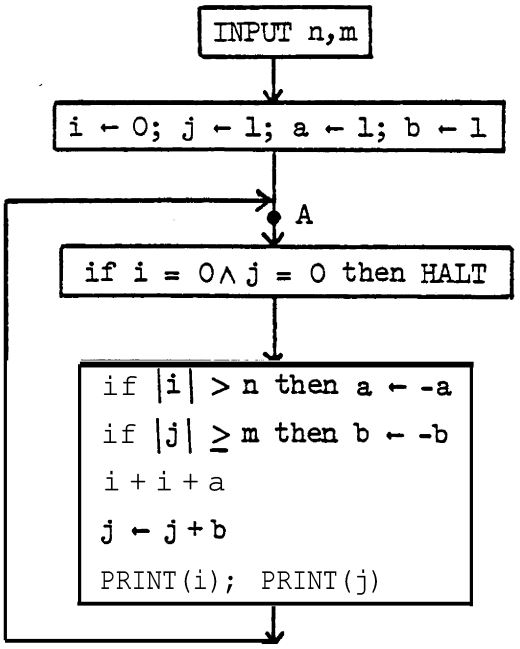
(b) by 3 [3 points]

(c) by 7 [7 points]

a regular set? Give arguments (but not necessarily machines or grammars) for your answers.

Problem 3. Program Verification. (12 points).

Consider the following flow-chart program:



Supply an inductive assertion for point A which is sufficient to demonstrate that the program does not terminate (regardless of the values of n, m) -- that is, an inductive assertion which implies that $i \neq 0 \vee j \neq 0$.

Problem 4. NP-completeness. (12 points).

Let $R_0, R_1, \ldots, R_n$ be a list of rectangles with positive integer length sides. Show that the following problem can be solved in non-deterministic polynomial time.

"Decide whether rectangles $R_1 \ldots R_n$ can be placed inside R_0 in such a way that

(1) No two R_i, R_j, $i \neq j$, $i,j > 0$ overlap, and

(2) The sides of the R_i are parallel to the sides of R_0."

(The rectangles are given by pairs of integers $\langle i_0, j_0 \rangle, \langle i_1, j_1 \rangle, \ldots, \langle i_n, j_n \rangle$, in binary notation, where i_k, j_k are the lengths of the sides of R_k.)

Problem 5. Decidability. (12 points).

(a) (6 points). Show that there exists a function f on the natural numbers which grows faster than any recursive function. That is, we want a function $f: N \to N$ such that for all recursive $g: N \to N$, $\exists n \forall (m > n)\,(f(m) > g(m))$.

(b) (6 points). Show that there exists an **infinite** set of natural numbers which has no infinite recursive subsets. (You may use the result of part (a) whether or not you were able to prove it.)

Problem 1.

```
i ← 1;
j ← n;
while x_i+y_j ≠ S and i < n and j > 1 do
if x_i+y_j > S then j ← j-1
              else i ← i+1;
if x_i+y_j := S then print (i,j);
             else print ("no such i and j")
```

This works because if there is a pair (i_0, j_0) such that $x_{i_0} + y_{j_0} = S$ then the variables i and j in the above algorithm have the property that $i \leq i_0$ and $j \geq j_0$. This is easily proved by induction. Since either i increases or j decreases at each iteration we know that within 2n iterations $i = i_0$ and $j = j_0$.

If there is no such pair (i_0, j_0), then the test $x_i + x_j \neq S$ will always be satisfied, and the loop will terminate when $i = n+1$ or $j = 0$, and the correct statement will be printed. This also happens in at most 2n iterations.

Problem 2.

(1) Suppose two children V_1 and V_2 had bad edges to their parent V. Because these edges are bad, we know that

$$2 \times (\text{weight of } V_1) > \text{weight of } V$$

$$2 \times (\text{weight of } V_2) > \text{weight of } V .$$

Adding these equations we get

$$\text{weight of } V_1 + \text{weight of } V_2 > \text{weight of } V .$$

But this is impossible since the weight of a node is at least the sum of the weight of its children.

(2) Consider a traversal up the tree from any leaf to the root. Each time we traverse a good edge the weight of the node we are at at least doubles. When we traverse a bad edge the weight cannot decrease. Since the leaves have unit weight, the number of time the weight can double is at most $\log_2 n$. Therefore there are at most $\log_2 n$ good edges along any path.

(3) We distinguish two types of nodes. Type 1: those that are at the root of a bad path (or not on a bad path at all), and Type 2: those that are inside a bad path. Aeach type 1 node we simple store the fact that it is a type 1 node, and a pointer to its parent. At each type 2 node we store (1) a pointer to the root of the bad path and (2) a level number which tells the distance from the node to the root of the bad path.

The algorithm traverses to the root from node x and node y following the pointers described in the previous paragraph. The sequence of nodes traversed are stored in two stacks. At the end, the top of both stacks contain the root of the tree. We now pop both stacks in unison until they show a different top element. If either of these elements are type 1, then select the parent of that one. If both of them are type 2, then select the one with a lower level number. The node selected is the least common ancestor of x and y . The running time is $O(\log n)$ since there are only $\log_2 n$ good edges along any path to the root.

Problem 3.

(a) We arrange them in correct order from right to left, first putting n in place, then n-1 . then 2 . Element 1 will then automatically be in place. To get j in place, we first flip it to the left, then flip it to its proper position. The total number of flips needed is thus $2(n-1)$.

(b) In the permutation 12 3 . .. n the number of adjacent elements that differ by one is n-1 . In the permutation

$2\ 4\ 6\ 8 \ldots \left(n - \frac{1}{2} + \frac{1}{2}\right)\ 1\ 3\ 5 \ldots \left(n - \frac{1}{2} + \frac{1}{2}\right)$ there are no adjacent elements differing by one for $n > 4$. Each flip can change the adjacency at only one place, therefore each flip can change the number of adjacent elements differing by one by at most one, and n-1 flips are needed to go between the two permutations given above. Q.E.D.

Problem 4.

Choose a set of edges which are a spanning tree of the graph. Label the vertices of the spanning tree with 1's and 0's such that the root is labelled 0 , and opposite ends of a spanning edge are labelled the same if the edge is + and different if the edge is - . This can be done in linear time with depth-first or breadth-first search. Now check, that each edge not in the spanning tree has the correct sign property, i.e., that opposite ends of a + edge are labeled the same and opposite ends of a - edge are labeled differently. The graph is balanced iff each edge has the correct sign property.

Proof: Consider a balanced graph. Any edge not in the spanning tree of the graph must have the correct sign property since the cycle formed by the edge and the spanning tree has an even number of - edges. To show the converse, note that in a graph with the correct sign property each cycle must have an even number of transitions from 1 to 0 or 0 to 1 because it starts and ends with the same label. Each such transition corresponds to a - edge, thus the number of - edges around any cycle must be even. Q.E.D.

Problem 1.

(a) The three types of PLANNER theorems are consequent theorems (THCONSE or IF-NEEDED) and two types of antecedent theorems (THANTE or IF-ADDED and THERASING or IF-REMOVED). All are invoked by pattern matching. An IF-NEEDED method is invoked in backward chaining when its pattern matches a subgoal. An IF-ADDED method is invoked whenever its pattern matches an assertion placed in the data base. An IF-REMOVED method is invoked whenever its pattern matches an assertion removed from the data base.

(b) MYCIN's backward chaining most closely resembles IF-NEEDED methods.

(c) HEARSAY-II's knowledge sources most closely resemble IF-ADDED methods (and possibly IF-NEEDED and IF-REMOVED).

Problem 2.

A variety of formulations are acceptable due to axioms about the relations involved. The following are just samples.

(a) $\exists x\ ON(BK,x) \wedge ROW(x,6) \wedge COL(x,5)$

(b) $\forall p \forall q \forall s\ ON(p,s) \wedge ON(q,s) \Rightarrow p = q \wedge \forall p \forall s \forall t\ ON(p,s) \wedge ON(p,t) \Rightarrow s = t$

(c) $P \in PAWN$
$\wedge\ WHITE(P)$
$\wedge\ \exists s\ ON(P,s) \wedge ROW(s,7)$
$\wedge\ \exists s \exists c \exists t\ ON(P,s) \wedge COL(s,c) \wedge ROW(t,8) \wedge COL(t,c) \wedge EMPTY(t)$

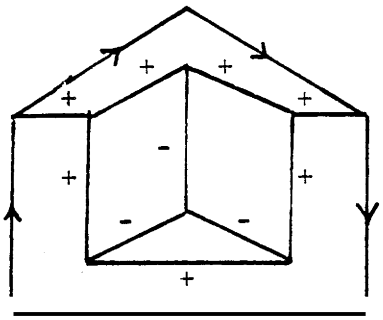
Problem 3.

(a)
WP-1 on A7
example WP-1 on B7 teaches column unimportant
example WP-2 on A7 teaches P = WP-1 unnecessary
near miss WB-1 on A7 teaches P ∈ PAWN necessary
near miss BP-1 on A7 teaches WHITE (P) necessary
near miss WP-1 on B6 teaches ROW 7 necessary
near miss WP-1 on A7
BR-1 on A8 teaches EMPTY destination necessary

(b) No, because there's no way to encode the constraint that the king and rook must not have moved. Another example would be the en passant

Problem 4.

(a)



(b)

	before H	before I
A	L1	L1
B	A1	A1
C	L1	L1
D	A1	A1
E	L1	L1
F	A2, A3	A3
G	A2, A3	A3
H		F1

Problem 5.

(a) The bishop beside the white pawn.

(b) The king pawn (noun modifiers not handled).

(c) Those pawn (number mismatch).

(d) The pawn (which pawn?)

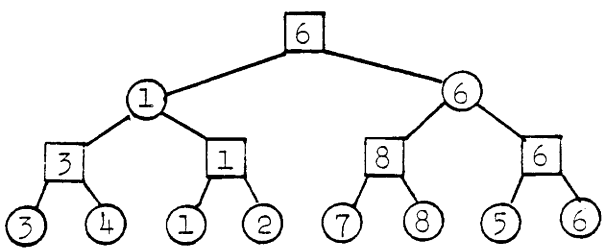
(e) The pawn on the square next to the king (attachment of the second prepositional phrase).

Problem 6.

(a) The Branch and Bound method does not use heuristic information.

(b) Depth-first search is the only method applicable.

(c) α-β search is not always more efficient than exhaustive minimax, though it's never worse. Consider the following example.



(d) No, the robot is not guaranteed to get the optimal result, since he may arrive at a ridge or foothill in picture quality space from which any adjustment degrades the picture.

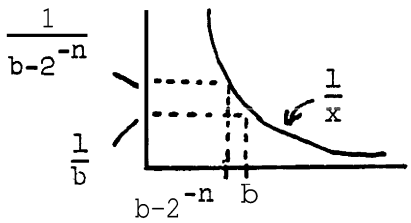
Problem 7.

Cognitive Simulation and Machine Intelligence.

Problem 1.

(a) This question asks for the range of values that the mantissa can take. If all the bits of the mantissa are 0 then 0 is represented. If there is at least one 1 in the mantissa then the range must be $1/8 \leq$ range < 1 .

(b)



We want to find out how close b and $b-2^{-n}$ have to be so that $1/b$ and $1/(b-2^{-n})$ are within 2^{-6} of each other.

The number of bits that b and $b-2^{-n}$ agree is the number of bits needed from the mantissa in order to lookup the reciprocal to an accuracy of 2^{-6}.

$$-\frac{1}{b} + \frac{1}{b-2^{-n}} < 2^{-6}$$

$$-b + 2^{-n} + b < 2^{-6}(b^2 - 2^{-n}b)$$

$$+ 2^{-n} < 2^{-6}b^2 - 2^{-6}2^{-n}b$$

ignore this term, too small

The most critical value of b is 1/8, since that is where the reciprocal function has its greatest effect.

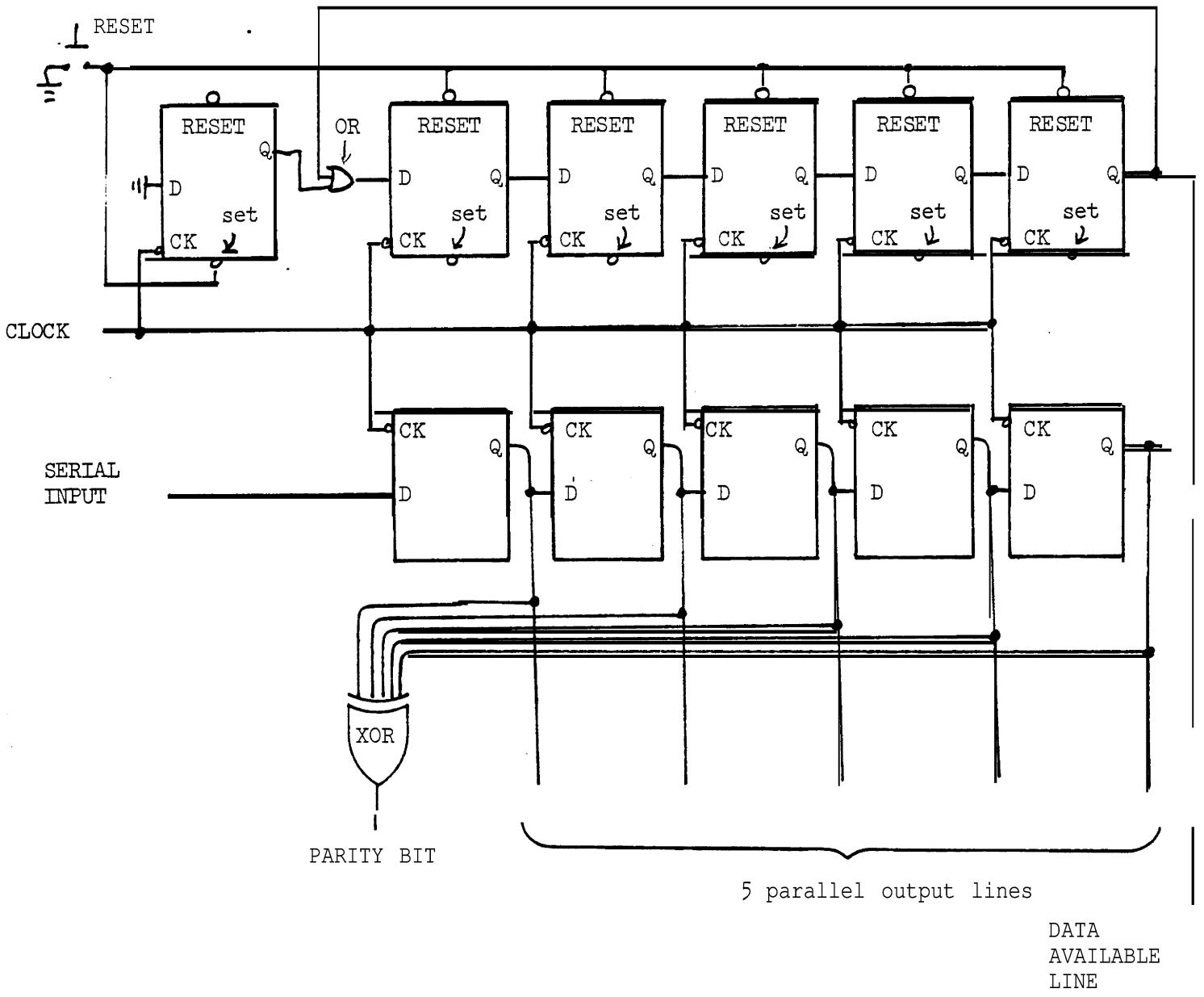
$$2^{-n} < 2^{-6}(2^{-3})^2$$

$$2^{-n} < 2^{-12}$$

or that $n > 12$ so use $n = 13$ bits.

(c) There are 2^{13} words in the table since one must use 13 bits to lookup the answer.

Problem 2.



The bottom row of D-FF's shift in the serial data, one bit for every clock pulse received. The top row of D-FF's act as a 5 bit circular shift register with one "1" cycling around along with 4 "0"'s. When the "1" reaches the last D-FF 5 clock pulses have gone by so that the bottom D-FF's must have shifted in 5 data bits.

The first D-FF on the top row is set to a "1" by RESET (all the rest of the top row is set to "0" by RESET) and it "injects" that "1" into the circular shift register on the next clock pulse.

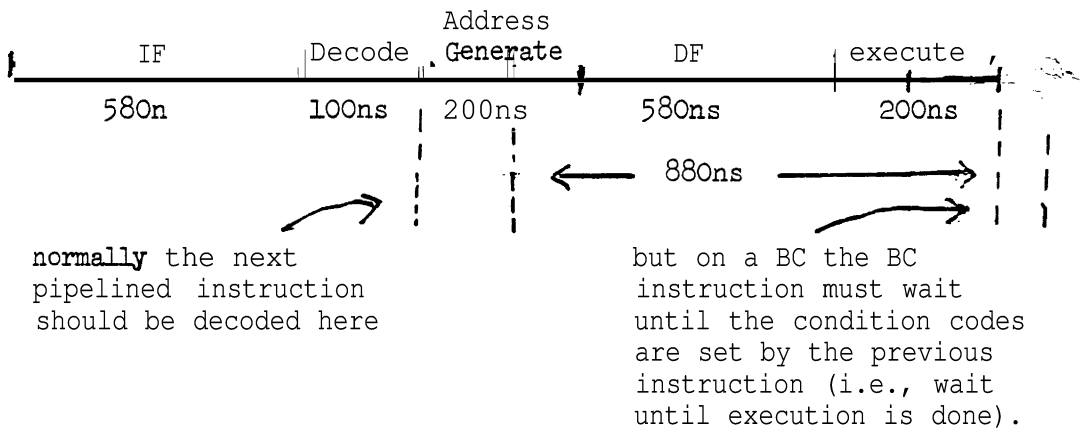
Problem 3.

(a) The request rate to memory is

(0.62 instr memory fetch/instr + 0.78 data memory fetches/instr)
* 10 million instr/second (CPU decodes 1 instr per internal cycle
= 100 ns))

+ 0.78) x 10^7 = 14×10^6 fetches/sec = 14 MAPS

(b) Peak performance is 100 ns/instruction = 10×10^6 instr/sec
= 10 MIPS



∴ The BC instruction causes a loss of 880ns compared to normal pipelining operation.

The "average" instruction time is

0.68 x 100ns (normal) + 0.32 x 980ns (BC instr)
≈ 382ns/instr ≈ 2.6 MIPS

Problem 4.

(a) The cache is a high speed memory between the CPU and main memory. It uses some algorithm to try to contain words of memory which it anticipates the CPU will ask for, thereby providing faster access to those words than main memory can achieve.

(b) A hard-wired CPU is generally faster but it is more complex to build and, once built, its operation cannot changed.
A micro-coded CPU is slower, is generally simpler to design and build and has the flexibility of changing its instruction set by changing the micro-code it contains.

(c) TTL -- high power, fairly fast logic. Is the standard for SSI and MSI chips.

CMOS -- very low power, rather slow, with very high noise immunity.

NMOS -- low power, moderately fast. Mostly used in LSI chips.

Problem 1.

(a) Consider the 2x2 example

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} \qquad \text{with } 0 < \epsilon << 1$$

with no partial pivoting we get

$$A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}$$

The growth factor is $1/\epsilon$ which can be arbitrarily large. This example can be easily extended to the $n \times n$ case.

(b) With no pivoting, after the k-th step of the elimination, A has become

$$A^{(k)} = \begin{bmatrix} \bar{a}_1 & \bar{b}_1 & & & & & \\ & \bar{a}_2 & \bar{b}_2 & & & & \\ & & \ddots & \ddots & & & \\ & & & \bar{a}_{k+1} & \bar{b}_{k+1} & & \\ & & & c_{k+2} & a_{k+2} & b_{k+2} & \\ & & & & \ddots & \ddots & \\ & & & & & c_n & a_n \end{bmatrix}$$

Let $\alpha = \max_i \{|a_i|, |b_i|, |c_i|\}$. Applying the next step of the elimination gives

$$\bar{b}_{k+2} = b_{k+2}$$

$$a_{k+2} = a_{k+2} - \frac{c_{k+2}}{\bar{a}_{k+1}} \bar{b}_{k+1} \quad .$$

Clearly $|\bar{b}_{k+2}| \leq \alpha$. Suppose inductively that

(1.1) $$|c_{k+2}| \leq |\bar{a}_{k+1}| \quad .$$

Then

(1.2) $$|a_{k+2}| - |\bar{b}_{k+1}| \leq |\bar{a}_{k+2}| \leq |a_{k+2}| + |\bar{b}_{k+1}| \quad .$$

Hence $|\bar{a}_{k+2}| \leq 2\alpha$ and so $G \leq 2$, provided we can show that (1.1) continues to hold, i.e., that

$$|c_{k+3}| \leq |\bar{a}_{k+2}| \quad .$$

From the diagonal dominance of A, we know that

$$|c_{k+3}| + |b_{k+1}| \leq |a_{k+2}| \quad .$$

so

$$|c_{k+3}| \leq |a_{k+2}| - |b_{k+1}|$$

$$\leq |\bar{a}_{k+2}| \qquad \text{from (1.2)}$$

Problem 2.

(a) Let $S_1^n = \sum_{j=1}^{n} x_j$ and suppose that

$$fl(S_1^{N-1}) = \sum_{j=1}^{N-1} x_j(1+\alpha_j) \quad \text{with} \quad |\alpha_j| < (N-1)u + O(u^2) \quad .$$

Then $fl(S_1^N) = \left[\sum_{j=1}^{N-1} x_j(1+\alpha_j)\right](1+\epsilon_1) + x_N(1+\epsilon_2)$ with $|\epsilon_i| < u$ for $i = 1, 2$. So

$$fl(S_1^N) = \sum_{j=1}^{N-1} x_j(1+\alpha_j+\epsilon_1+\alpha_j\epsilon_1) + x_N(1+\epsilon_2) \quad .$$

Let

$$\beta_j = \alpha_j + \epsilon_1 + \alpha_j\epsilon_1 \qquad \text{for } j = 1,2,\ldots,N-1$$

$$\beta_N = \epsilon_2 \quad .$$

Then

$$fl(S_1^N) = \sum_{j=1}^{N} x_j(1+\beta_j)$$

and

$$|\beta_j| \leq |\alpha_j| + |\epsilon_1| + |\alpha_j\epsilon_1|$$

$$< (N-1)u + u + O(u^2) = Nu + O(u^2)$$

for $j = 1,2,\ldots,N-1$ and

$$|\beta_N| < u \leq Nu$$

which completes the proof by induction.

(b) Let $S_2^k = \sum_{j=1}^{2^k} x_j$, $\tilde{S}_2^k = \sum_{j=1}^{2^k} x_{2^k+j}$ and suppose that

$$fl(S_2^k) = \sum_{j=1}^{2^k} x_j(1+\alpha_j)$$

$$fl(\tilde{S}_2^k) = \sum_{j=1}^{2^k} x_{2^k+j}(1+\alpha_{2^k+j})$$

with $|\alpha_i| < ku + O(u^2)$ for $i = 1,2,\ldots,2^{k+1}$. Then

$$fl(S_2^{k+1}) = \left[\sum_{j=1}^{2^k} x_j(1+\alpha_j)\right](1+\epsilon_1) + \left[\sum_{j=1}^{2^k} x_{2^k+j}(1+\alpha_{2^k+j})\right](1+\epsilon_2)$$

with $|\epsilon_i| < u$. Let

$$\gamma_j = \alpha_j + \epsilon_1 + \alpha_j\epsilon_1 \qquad \text{for } j = 1,2,\ldots,2^k$$

$$\gamma_j = \alpha_j + \epsilon_2 + \alpha_j\epsilon_2 \qquad \text{for } j = 2^k+1,\ldots,2^{k+1}$$

Then

$$fl(S_2^{k+1}) = \sum_{j=1}^{2^{k+1}} x_j(1+\gamma_j)$$

and

$$|\gamma_j| < ku + u + O(u^2)$$
$$= (k+1)u + O(u^2) .$$

Problem 3.

a) It follows from the mean value theorem that

$$\frac{f(x_*) - f(x_n)}{x_* - x_n} = f'(\xi_n)$$

for $\xi_n \in [\min x_*, x_n, \max(x_*, x_n)]$. Since $f(x_*) = 0$ we have

$$x_* - x_n = -f(x_n) / f'(\xi_n) \tag{3.1}$$

if $f'(\xi_n) \neq 0$. Using Newton's method the $(n+1)$-st iterate is given by

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

and consequently

$$(3.2) \qquad |X_{n+1} - x_n| = |f(x_n)/f'(x_n)| .$$

Since ξ_n is between x_n and x_* and $x_n \to x_*$, we can expect $f'(x_n)$ to be a good approximation of $f'(\xi_n)$. From (3.1) and (3.2) we obtain

$$|x_* - x_n| = |f(x_n)/f'(\xi_n)| \approx |f(x_n)/f'(x_n)| = |x_{n+1} - x_n| .$$

Thus $|x_{n+1} - x_n|$ is usually a good error estimate to use as a termination criteria for a Newton iteration.

(b) Utilizing the estimate of (a) above and the table of values of r_n we see that $p = 1$ and $c \approx 2/3$. Since Newton's method converges linearly to multiple roots with asymptotic error constant (AEC) $(m-1)/m$ for roots of multiplicity m, it is reasonable to conjecture that our iterates are converging to a root of multiplicity 3 $((3-1)/3 = 2/3)$. If this is the case and we replace $f(x) = 0$ by $f'(x) = 0$ we would improve the AEC, it would become $1/2$, but the order p will still be one. Using $f''(x) = 0$ the root becomes simple and the convergence quadratic, $p = 2$.

Problem 4.

We have

$$\begin{aligned} y_n &= f(y_{n-1}) + \epsilon_n \\ &= f(f(y_{n-2}) + \epsilon_{n-1}) + \epsilon_n \\ &\vdots \\ &= f^{(n)}(y_0) + f^{(n-1)}(\epsilon_1) + \cdots + f(\epsilon_{n-1}) + \epsilon_n \end{aligned}$$

where $f^{(n)} = f \circ f \circ \cdots \circ f$ (n occurrences of f) and

$$X_n = f^{(n)}(x_0) .$$

Consequently,

$$|x_n - y_n| \le |f^n(x_0) - f^n(y_0)| + \sum_{j=0}^{n-1} |f^{(j)}(\epsilon_{n-j})|$$

and using Lipschitz continuity

$$\le K^n |x_0 - y_0| + \max_j |\epsilon_j| \sum_{j=0}^{n-1} K^j$$

$$= K^n |x_0 - y_0| + \max_j |\epsilon_j| \frac{K^n - 1}{K - 1} .$$

Computer Lanquage Syntax

1a. [4**2**3 = 4^{2^3} = 65536 rather than $(4^2)^3$ = 4096]

1b. <expr> ::= <expr> + <term> |<expr> - <term> | <term>

<term> ::= <term> * <factor> |<term>| <factor> |<factor>

<factor> ::= <unsfact>|- <unsfact>; don't recurse here!

<unsfact> ::= <base> ↑ <exp>| <base>

<base> ::= (<expr>) | <identifier>

<exp> ::= <factor>; or less fancy:

<exp> ::= <unsfact>

Pagiq

2a. DPA selects pages from the entire real space for removal, WSA manages pages for individual user spaces.

2b. Control over total page demand to assure that the residence time of a page is sufficient to allow its use, relative to its acquisition cost.

2c. Good estimates of WS's and control over number of users to keep aggregate memory demand for WS's within real bounds. WS is determined by a time window

Cooperating Processes

```
3.  User ui : FOR EVER
                BEGIN
                   uk = AWAIT (ui, mess);
                   process;
                   SEND (ui, uj, mess)
                END

    Bufferhandler : MONITOR
        DECLARE bufferspace, source list, dest list, bufferpointers, state;
        PROCESS SEND (ul,u2,m);
            bp := acquirebuffer; if bp = null {put ul to sleep; exit);
complete:send sourcelist (bp) := ul; destlist (bp) := u2;
            bufferspace (bp) := m;
            Wakeup (u2); wakeup waiting u processes;
            Return
            END SEND:

        PROCESS AWAIT (u,m)
           op := locate buffer-for (u); if none {put u to sleep; exit};
           m := bufferspace (op);
           dest := sourcelist (op);
           wasbusy := release-space (op); → IF wasbusy complete;
           return (dest)
           end AWAIT;
```

Cooperating Processes (continued)

```
INTERNAL PROCESS ACQUIREBUFFER;
     If full Estate := busy, return (null)};
     else find bp;
     return (bp)
     END ACQUIRE-BUFFER ;

INTERNAL PROCESS RELEASESPACE (p)
     Mark p free;
     state := not busy;
     return (pstate)
     END RELEASE SPACE ;

Initialize:  Set bufferpointers, state

END buffer handler .
```

Computer Lanquages

4a. Little support code at run-time is required, and execution can be more efficient. The former, in-turn, reduces machine size requirements and improves compiler consistency and portability.

4b. Parameters are inflexible, strings are primitive, arrays cannot be adjusted, file access is limited, garbage is not being collected.

Language System

5 i. Heap just grows with every NEW statement.
Problem: can run out of space, unsuitable for major systems.

ii. Heap is managed as stack, a DISPOSE function allows popping of the stack.
Problem: User has to understand and use the mechanism correctly, can loose records he is still using.

iii.. Records are linked back to all of their references, dispose checks all or selected records for safe removal. Heap space contains a free list or is compressed as needed.
Problem: complex mechanism, difficulty in releasing unused circular ring structures.

iv. Garbage Collection is applied to the heap.
Problem: Pointers and record boundaries have to be recognizable, imposes constraints on code generation and requires extra space. The lack of a single structure root in PASCAL records makes circular structures hard to collect. Compression is very difficult.

Problem 1.

(a)

	T	F
T	T	F
F	F	T

(i.e., the standard interpretation of " ≡ ").

(b) $P \supset (P \supset P)$.

(c) Since the axioms and rules are sound for the nonstandard interpretation for " $\supset$ ", it follows by induction on the length of proofs that every formula provable in the given system is valid for the nonstandard interpretation. $P \supset (P \supset P)$ is not valid for the nonstandard interpretation (take $P = F$), and therefore is not -provable,

Problem 2.

The set S_k of decimal representations of positive integers divisible by any positive integer k is regular. Proof: S_k is recognized by the following automaton, R_k. R_k has k states $q_0, \ldots, q_{k-1}$. There is a transition from state q_i to state q_j; labeled n iff $((i * 10)+n) \bmod k = j$. q_0 is the initial state, and the only accepting state. (At each stage, R_k is in state i iff the part of the number read so far is equal modulo k to i .)

Problem 3.

" $i + j$ is odd, and $|a| = 1$, and $|b| = 1$."

Problem 4.

The problem is in NP, since, if $R_1 \ldots R_n$ can be placed into R_0 without overlap, then the R_i can be placed in R_0 without overlap in such a way that the coordinates of the lower left hand corners of all of the R_i are integers. This is shown as follows. Suppose that there is a legal placement of the R_i in R_0. Consider the leftmost block whose horizontal coordinate is not an integer. This block can be moved over, without overlap, to a position with integer horizontal coordinate. Repeat until all blocks have integer horizontal coordinates. Do the same for vertical coordinates.

To determine whether there is a legal placement of the R_i in R_0 in nondeterministic polynomial time, guess orientations. and integer coordinates for the blocks. Overlap can then be checked in polynomial time.

Problem 5.

(a) Let $g_0, g_1, \ldots$ be an enumeration of the recursive functions. Take

$$f(n) = \max_{i<n} (g_i(n)) + 1 .$$

Then, for each n , $\forall(m \geq n)\,(f(m) > g_n(m))$.

(b) $S = \{n \mid \exists i(f(i) = n)\}$ has the desired property. Suppose $R \subseteq S$ is recursive and infinite. Define g(n) = the n-th member of R when R is listed in increasing order. Then g(n) is a recursive function, but, g grows more rapidly than f . Contradiction.

Winter 1980 - Comprehensive Programming Project

Display of Mathematical Expressions

The goal of this problem is a program that takes as argument a list structure representation of a mathematical expression in prefix form and prints it in standard two-dimensional format. For example, the expression

(plus (expt x 2) (times 2 x y) (expt y 2))

should be displayed as

$$x^2 + 2XY + Y^2$$

The problem is made more difficult by the possibility that the expression cannot fit on a single line. It is okay to "break" an expression across lines before sum and product operators, provided those operators are not embedded within exponential expressions or quotients. Consider, for example, the following expression.

$$1 + \frac{\text{OMEGA TIME}}{2} + \frac{\text{OMEGA TIME}^2}{6} + \frac{\text{OMEGA TIME}^3}{24} + \frac{\text{OMEGA TIME}^4}{120}$$

$$+ \text{OMEGA TIME}^5 - w - p - - + \frac{\text{CNEGA TIME}^6}{51348} + m - - M - - - + \frac{\text{CNEGA TIME}^8}{362886:}$$

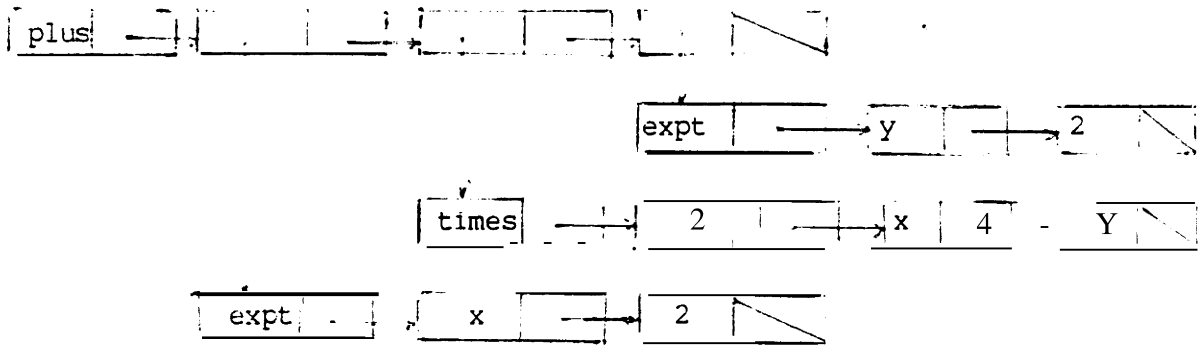
Obviously, such expressions can be broken at a variety of places. Your program should choose a display format that minimizes the total number of breaks subject to the constraint that at least half of every line is utilized (whenever the expression is more than a half line long).

In your implementation you may use Interlisp, Maclisp, Algol W, Pascal, or Sail. Your efforts will be graded according to the criteria of correctness, clarity, efficiency, and documentation.

Inputting Expressions

Expressions are built from integers and alphanumeric identifiers that begin with alphabetic characters. You need consider only four operators, viz. addition (plus), multiplication (times), division (quot), and exponentiation (expt) .

Expressions should be represented in LISP-like list format. For example, the first expression above would have the following box and pointer structure.



Write functions ADD, MUL, DIV, and POW that take expressions as arguments and produce the corresponding list structure. For example, to create the above expression, one would write the following (in LISP) .

(ADD (POW 'X 2) (MUL 2 'X 'Y) (POW 'Y 2))

While DIV and POW are inherently binary operators, you may wish to write ADD and MUL as n-ary functions. In some languages, it is not possible to define subroutines with an arbitrary number of arguments. If you select one of these languages, you may write several series of functions (ADD2, MUL2, ADD3, MUL3, . . .) that take the number of arguments indicated in the name. For example, the PASCAL version of the above expression would be as follows.

ADD3 (POW ("X","2"),MUL3("2", "X", "Y"), POW("Y","2"))

Display Format

Displaying expressions tastefully is an art with a large number of conventions. For the purposes of this problem, your program need use only the following rules.

1. Sums should be displayed with an infix " + ". (That's space, plus, space.)

2. Products should be displayed with an infix space or "*".

3. Quotients should be displayed as numerator over denominator, with the quotient bar made of dashes occupying the line between the two. The numerator and denominator should be centered within the quotient.

4. Exponential expressions should be displayed with raised exponents, in which the lowest line of the exponent is one above the highest line of the base (with the exception noted below) .

5. Parentheses should be inserted where necessary to avoid ambiguity, e .g .

$$A^{2} \qquad A\,(-)\,B \qquad A+B \qquad (A+B)^{2} \qquad X^{X} \qquad (X^{X})^{X}$$

Note that after a closing parentheses, an exponent is lowered to the line above the parentheses. Obviously, these are not the only cases of ambiguity.

Breaking Lines

The constraints on breaking expressions across lines are as given in the introduction. One good approach is to put as much on a line as will fit, then go on to the next line. Unfortunately, this will sometimes fail to satisfy the half-line requirement, as in the following example.

$$\text{OMEGA T} - + \frac{\text{CMEGAT}^{22}}{R} + \frac{\text{CMEGAT}^{33}}{R} + - - - - + \frac{\text{OMEGA T}^{44}}{R} + \frac{\text{OMEGA T}^{55}}{R} + - - - - \frac{\text{OMEGA T}^{66}}{R}$$

$$+ \frac{\text{OMEGA T} + \text{OMEGA T}^{22}}{R^2}$$

$$+ \frac{\text{OMEGA T} + \text{OMEGA T}^{22} + \text{OMEGA T}^{33} + \text{OMEGA T}^{44} + \text{OMEGA T}^{55}}{R}$$

In such situations it is necessary to try a different set of breakpoints. Just taking a fragment from the previous line will sometimes work, but the effect may propagate. In general, a combinatorial search is required. In implementing an efficient search, you may find dynamic programming techniques helpful.

There are some expressions for which no display can be generated that satisfies the given constraints. If the half line constraint cannot be satisfied, your program should choose a set of breakpoints that minimizes the total number of lines. If the input contains a fragment that does not fit on a single line and cannot be broken (e.g. a quotient), you may have your program do whatever you wish (e.g. print an error message, write quotients in infix form (A/B) with a breakpoint at the slash, etc.).

Hints

o You should structure your program into three distinct parts:

1. a "dimensioning" subroutine, which determines the size of subexpressions and makes a list of possible breakpoints

2. a program that decides which breakpoints to use

3. a program that prints the expression on the terminal

The graders will look for these three subparts in examining your work.

o This is a "data structure intensive" project. Choose good data structures and document their significance carefully.

o To facilitate testing your program, use a global parameter called LINELENGTH whose value determines how wide an expression may be.

Test Data

Test data for your program will be given in two sets. The first set is attached here. The second set will be available after 9:00 a.m. on Monday, January 21, in Jacks 206. Turn in your source program, associated documentation, and the output from all the test data.

Questions

If you have questions about the problem, you may contact

Rod Brooks	ROD@SAIL	497-1604	856-0979

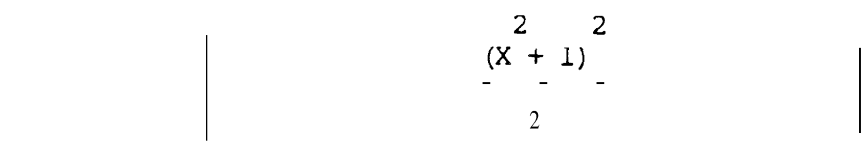
or

Mike Genesereth	GENESERETH@SUMEX	497-3728

Winter 1980 - Comprehensive Programming Project

Test Data I

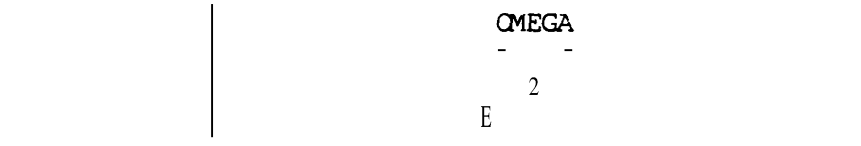
1. (SETQ LINELENGTH 40.)
   (DISPLAY (DIV (POW (ADD (POW 'X 2) 1) 2) 2))



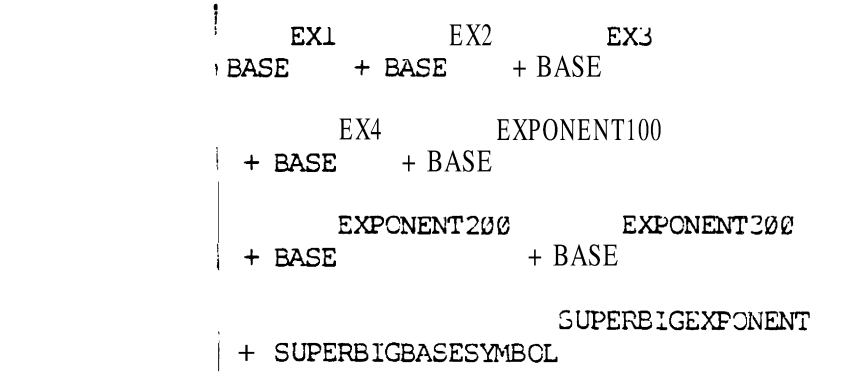
2. (DISPLAY (POW 'X (POW 'X 'X)))

3. (DISPLAY (POW (POW 'X 'X) 'X))

4. (DISPLAY (POW 'E (DIV 'OMEGA 2)))



5. (DISPLAY (ADD (POW 'BASE 'EX1)
                 (POW 'BASE 'EX2)
                 (POW 'BASE 'EX3)
                 (POW 'BASE 'EX4)
                 (POW 'BASE 'EXPONENT100)
                 (POW 'BASE 'EXPONENT200)
                 (POW 'BASE 'EXPONENT300)
                 (POW 'BIGBASESYMBCL 'SUPERBIGEXPUNENT)))



Test Data I1

1. (SETQ LINELENGTH 40.)
   (DISPLAY (POW (DIV (ADD (POW 'X 4) (POW 'X 3) (POW 'X 2))
   2)
   2))

$$\left(\frac{x^4 + x^3 + x^2}{2}\right)^2$$

2. (DISPLAY (DIV 'A (DIV 'B 'C)))

$$\frac{A}{\left(\frac{B}{C}\right)}$$

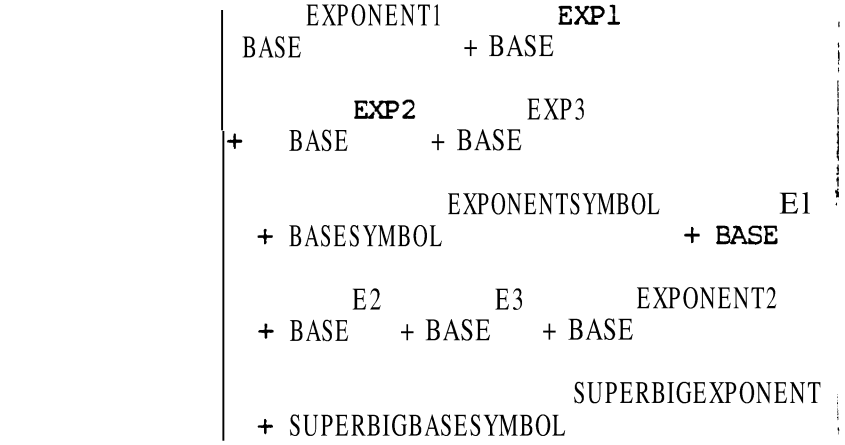
3. (DISPLAY (ADD 'A (ADD 'B 'C) 'D))

$$A + (B + C) + D$$

4. (DISPLAY (ADD (POW 'SUPERBIGBASESYMBOL 'SUPERBIGEXPONENT)
   (POW 'BASE 'EXPONENT300)
   (POW 'BASE 'EXPONENT200)
   (POW 'BASE 'EXPONENT100)
   (POW 'BASE 'EX4)
   (POW 'BASE 'EX3)
   (POW 'BASE 'EX2)
   (POW 'BASE 'EX1)))

$$SUPERBIGBASESYMBOL^{SUPERBIGEXPONENT} + BASE^{EXPONENT300} + BASE^{EXPONENT200} + BASE^{EXPONENT100} + BASE^{EX4} + BASE^{EX3} + BASE^{EX2} + BASE^{EX1}$$

5. (DISPLAY (ADD (POW 'BASE 'EXPONENT1)
(POW 'BASE 'EXP1)
(POW 'BASE 'EXP2)
(POW 'BASE 'EXP3)
(POW 'BASESYMBOL 'EXPONENTSYMBOL)
(POW 'BASE 'E1) (POW 'BASE 'E2) (POW 'BASE 'E3)
(POW 'BASE 'EXPONENT2)
(POW 'SUPERBIGBASESYMBOL 'SUPERBIGEXPGNENT)))



6. (DISPLAY (DIV (ADD (POW 'E (TIMES 2 'OMEGA))
(POW 'E (TIMES 3 'OMEGA))
(POW 'E (TIMES 4 'OMEGA))
(POW 'E (TIMES 5 'OMEGA)))
2))

ERROR - Expression too wide

7. (DISPLAY (POW (ADD (POW 'E (TIMES 2 'OMEGA))
(POW 'E (TIMES 3 'OMEGA))
(POW 'E (TIMES 4 'CMEGA))
(POW 'E (TIMES 5 'OMEGA)))
2))

ERROR - Expression too wide

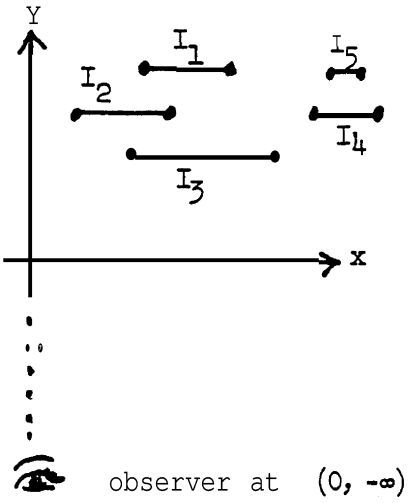
8. (DISPLAY (POW 'E
(ADD (POW 'E (TIMES 2 'OMEGA))
(POW 'E (TMES 2 'OMEGA))
(POW 'E (TIMES 4 'OMEGA))
(POW 'E (TIMES 5 'OMEGA)))))

ERROR- Expression too wide

Analysis of Algorithms

Problem 1. [18 points]

Consider a set of n disjoint line segments lying parallel to the x-axis. When one stands at $(0,-\infty)$ and looks toward the segments; some of them may not be seen. For example, segments I_1 and I_5 cannot be seen in Figure 1 (11 hides behind $I_2 \cup I_3$) .



(a) [6] Give an algorithm that, for any input set of segments $S = \{(a_1,b_1;y_1), (a_2,b_2;y_2), \ldots, (a_n,b_n;y_n)\}$, computes the subset $S' \subseteq S$ of those segments that cannot be seen. Note the triplet $(a_i,b_i;y_i)$ represents the line segment connecting the points (a_i,y_i) and (b_i,y_i) .

(b) [6] Give an algorithm that runs in time $O(n \log n)$.

(c) [6] Suppose the segments do not necessarily lie parallel to the x-axis, Give an O(n log n) -time algorithm for computing the hidden line segments. (See Figure 2.)

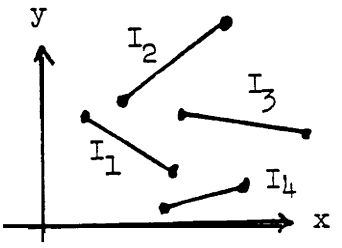


Figure2

Problem 2. [18 points]

Let $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_n$ be two binary strings of length n . A composition of A and B is a string of length 2n obtained by merging A, B in any manner. For example 011001 is a composition of 010 and 101 (the underlined part is 010 and the rest is 101).

(a) [12] Give a polynomial-time algorithm that determines if C is a composition of A and B , given the input strings A, B and C .

(b) [6] Make your algorithm run in time $O(n^2)$.

Problem 3. [24 points]

One of the more bizarre proposed solutions to last fall's 'bicycle crisis" was to have a row of n "sheds", numbered 1,2,...,n , each of which could hold one bicycle. Sheds have closed doors, and you cannot see into one without (temporarily) opening its door. These would be shared by m > n people, no more than n of which would be at work at one time. When arriving at work on your bicycle, you would open shed doors according to some agreed-upon strategy, eventually finding an empty shed (not necessarily the first empty one encountered), placing your-bicycle there, and closing the door.

On leaving work, you would open doors until you found your bicycle (which need not be in the shed you left it in); possibly you would rearrange some other bicycles, and then you would leave.

Formally, an algorithm consists of a sequence of one or more steps. A step consists of the following operations for some i :

(1) Open the door of shed i .
(2) If you have a bicycle with you, and shed i is empty, you may place that bicycle in shed i .
(3) If you have a bicycle with you, and shed *i* has a bicycle, you may exchange bicycles.
(4) If you have no bicycle, and shed i has a bicycle, you may take that bicycle from the shed.
(5) Close the door to shed i .

In operations (2),(3), and (4), the decision whether to manipulate bicycles can depend only on whether the bicycle you hold is your own, or whether the bicycle you see in the shed is your own, i.e., you can recognize your own bicycle, but cannot distinguish others.

A poor strategy we could follow is for each person, on arriving, to examine all sheds 1,2,...,n until an empty one is found, place his bicycle there, and remember the number of the shed used, Then, on leaving, since no one ever moves bicycles, simply go to that shed and remove the bicycle. This method requires O(n) steps on arrival and O(1) steps on leaving.

The problem is to devise another strategy, to be followed by all users of the sheds, that requires only $O(\sqrt{n})$ steps on arriving and leaving. An informal description of how to choose the sheds i on which steps are performed, together with how the decisions of operations (2)-(4) of these steps are made, will be acceptable.

Problem 1. [10 points]

Answer the following questions.

(a) [2] What is the advantage of a semantic network over the "record-oriented" representations described in data base management research in which each row represents an entity and each column represents an attribute, i.e., essentially tables.

(b) [2] What is the difference between backward chaining and GPS?

(c) [3] What is meta-planning? State at least two meta-planning heuristics.

(d) [3] Draw a small (and-or) search tree with branching factor of 2 and depth 4 . Assign values to the leaf nodes so that the α-β search algorithm searches as little as possible.

Problem 2. [10 points] Predicate Calculus.

(a) Encode the following sentences as formulas in the predicate calculus.

(1) [2] "The postman's brown hat is at the cleaner's."

(2) [2] "Everybody loves somebody." (2 meanings)

(3) [2] "There are 23 postmen in Palo Alto."

(b) Find the most general unifier for the following sets of formulas, where lower case letters signify variables.

(1) [2] P(u,v,A) , P(f(x),w,x) , P(f(y),z,z) .

(2) [2] P(f(x),x) , P(z,f(z)) .

Problem 3. [20 points]

One aspect of quizmanship is selecting a problem to work on next. On many exams you are instructed to look over the problems before starting in order to help in making this judgment. As ah example, consider how one might formulate a plan for taking the comprehensive, i.e., determining which order to work on the problems. You may assume that students are given point values for each problem and are told there will be no partial credit.

(a) [5] Arthur, a Master's student, must maximize his overall score and realizes that choosing the optimal subset of problems to work on is a bin packing problem. Moreover, he is not sure he can accurately guess how long each problem will take; and so he decides to use hillclimbing to search the tree of possible orderings. What simple measure should he use in conducting this search? Is the resulting plan optimal?

(b) [5] Beatrice is a Ph.D. candidate and must demonstrate **minimal** competency of (let's say) 20 points in each area. She decides to use a two phase approach. In the first stage, she strives for 20 points in each area using minimal time; then she tries to maximize her overall score. What formal search method should she use in the first stage and what are the relevant cost and/or heuristic functions?

(c) [5] Carleton has already passed the exam and is taking it again for the fun of it. He boasts that he can always predict precisely how long it will take him to solve a problem and complains that Beatrice's two-phase strategy is non-optimal. Is he right? If so, why? If not, show a counterexample.

(d) [5] Suppose we wanted to build a robot able to decide what strategy to use. What knowledge would the robot have to have?

Problem 4 [20 points]

A particular computer implementation of the Blocks World offers two commands to the user. **PUTON(x,y)** gets x onto y so long as x is a moveable object, y is a brick or the table, and both have clear tops, **PAINT(x,c)** changes x's color to c , provided that x is directly on the table. Use the following vocabulary in answering the questions below.

BLOCK -- set of all manipulable objects.

BRICK, PYRAMID, WEDGE -- sets of objects with the corresponding shapes.

B1, B2, P1, P2, W1, W2, TABLE -- objects.

CLEAR -- one place relation signifying a clear top.

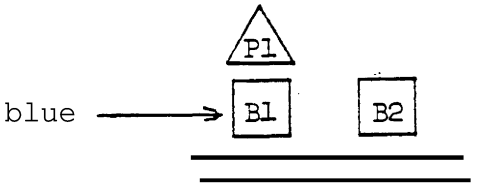
ON -- two place relation, a subset of BLOCK X(BRICK∪ {TABLE)) .

COLOR -- function from objects to hues.

RED, GREEN, BLUE, YELLOW -- hues.

(a) [3] Write STRIPS-like prerequisite, add, and delete lists for **PUTON.**

(b) [4] What sequence of actions would be required to achieve the goal (AND (ON B1 B2)(COLOR B1 RED)) in the following situation:



(c) [5] Suppose a robot using the Winston concept formation algorithm wanted to learn about PUTON by looking at examples of its operation. Show a sequence of block configurations that would teach the robot the prerequisites of PUTON.

(d) [5] What could go wrong? Present a legal training sequence that might give the robot an incorrect model of PUTON's prerequisities.

(e) [4] Suppose the robot used backward chaining and means-end analysis to solve problems. Devise a test problem to detect the error produced by the sequence in part (d).

1. Subroutine Linkage Architecture (12)

   a. (3) What machine registers are affected by CALL to a subprocedure?

   b. (4) How do you best transmit

      i) simple (i.e., single word) parameters
      ii) complex (i.e., record or array) parameters

   c. (5) Discuss the effect of recursive calls in these linkages.

2. High Speed Arithmetic (8)

   Discuss how pipelining complicates the handling of floating point exceptions (i.e., overflow, underflow, . ..).

3. Floating-Point Representation (15)

   The following table contains real numbers and their (octal representations in an 18-bit floating-point scheme. Describe this particular floating-point scheme.

-4	615777	1/4	172000
-3	601777	1/3	172525
-2	603777	1/2	174000
-1	605777	213	175252
1	202000	$2^{20}(2^{12}-1)$	377777
2	204000	$-2^{20}(-2^{12}-1)$	770000
3	206000	2^{-32}	002000
4	212000	-2-32	405777

5. Communication Alternatives (5)

   Describe the difference between synchronous and asynchronous communication protocols.

5. Combinational Circuit Problem (10)

The boolean Fibonacci function f is definded to be TRUE for the binary inputs x_1, x_2, x_3, x_4 corresponding to El, 2, 3, 5, 8, 131 and FALSE otherwise.

a. (5) Give the minimal sum of products expression for f.

b. (5) Give the minimal product of sums expression for f.

6. Circuit (10)

Sketch a circuit to implement the following precedence function.

X means don't care.

INPUTS				OUTPUTS			
A	B	C	D	PA	PB	PC	PD
1	x	x	x	1	0	0	0
0	1	x	x	0	1	0	0
0	0	1	x	0	0	1	0
0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0

Numerical Analysis

1. (10 points) Gaussian Elimination

A. (3 points) Suppose $A = LU$ are $n \times n$ matrices with L unit lower triangular and U upper triangular. Given L and U , how can $\det(A)$ be obtained?

B. (4 points) How many multiplications and divisions are required to obtain $\det(A)$ by the method of part A ? Include the cost of factoring A . Give the leading term.

C. (3 points) How many multiplications are required by the obvious expansion by cofactors method?

2. (20 points) Sparse Cholesky Factorization

A. (13 points) Suppose A is a symmetric positive definite $n \times n$ matrix. The rows of A contain leading sequences of zeros; in fact

$$a_{ij} = 0 \quad \text{for all } 1 \leq j < f_i \leq i ,$$

$$1 \leq i \leq n .$$

The numbers f_i are known in advance. Give a pseudo-algol procedure for computing the Cholesky (LL^T) factorization of A which makes as much use as possible of the structure of A .

B. (3 points) Give a multiplication count for your method.

C. (4 points) Suggest an appropriate data structure.

3. (15 points) Nonlinear Equations-.

Design a subroutine to compute $\sqrt[3]{c}$. Assume the computer uses rounded binary floating point arithmetic with a 24 bit fraction. The routine should use a rapidly convergent iterative method and stop when the approximation x satisfies

$$\frac{x - \sqrt[3]{c}}{\sqrt[3]{c}} \leq 2^{-20}$$

You should consider:

(i) reducing the range of values of C without introducing rounding error;

(ii) generating a good initial guess;

(iii) estimating the number of iterations required.

The method should use little storage; large tables are prohibited

4. (15 points) Polynomial Approximation

(3 points)

A. Let $r(x) = x^3 - \frac{3}{4}x$. What is $\|r\| \equiv \sup_{-1 \leq x \leq 1} |r(x)|$?

(6 points)

B. Show that if q(x) is a polynomial of degree ≤ 2 then

$$\|x^3 - q(x)\| \geq \|r(x)\| .$$

You may use:

Theorem: if $f \in C[-1,1]$ and $p(x)$ is a polynomial of degree n such that $f(x) - p(x)$ achieves its maximum values $\pm \max_{-1 \leq x \leq 1} |f(x) - p(x)|$ with successively alternating signs at n + 2 or more points in $[-1,1]$, then for any polynomial q(x) of degree < n

$$\|f - p\| \leq \|f - q\| .$$

(6 points) Prove the theorem,

Problem 1. Recursion Theory. [15 points]

Let x be a real number with $0 \leq x \leq 1$. We say that x is a "recursive real" if there exists a recursive function f mapping the non-negative integers into {0,1} which gives a binary expansion for x (i.e., $x = \sum_{i=0}^{\infty} f(i)2^{-i}$). Question: If x and y, with $0 < x < 1$, $0 \leq y < 1$, $x+y \neq 1$, are recursive, must x+y also be recursive? Justify your answer.

Problem 2. Logic. [15 points]

The compactness theorem for the predicate calculus with equality is as follows:

Theorem: Let S be an infinite set of sentences of the predicate calculus with equality. Suppose that every finite subset of S has a model. Then S has a model.

Use the compactness theorem to show the following: If φ is a sentence of the predicate calculus with equality such that for each number n there is a finite model of φ with more than n elements, then φ has an infinite model.

(Note: A model for a theory in the predicate calculus with equality must interpret the equality relation as identity in the model.)

Problem 3. Automata and Languages. [15 points]

Let G be a context-free grammar with n productions, each of which has at most m letters on its right hand side. Suppose that the language generated by G consists of just one sentence S. Give a tight upper bound for length of S in terms of n and m. Justify your answer.

Problem 4. NT-completeness. [15 points]

The firehouse problem is: given an undirected graph G, a maximum distance d, and a number of firehouses k, can you choose k vertices as firehouses so that every vertex is within distance d of at least one firehouse.

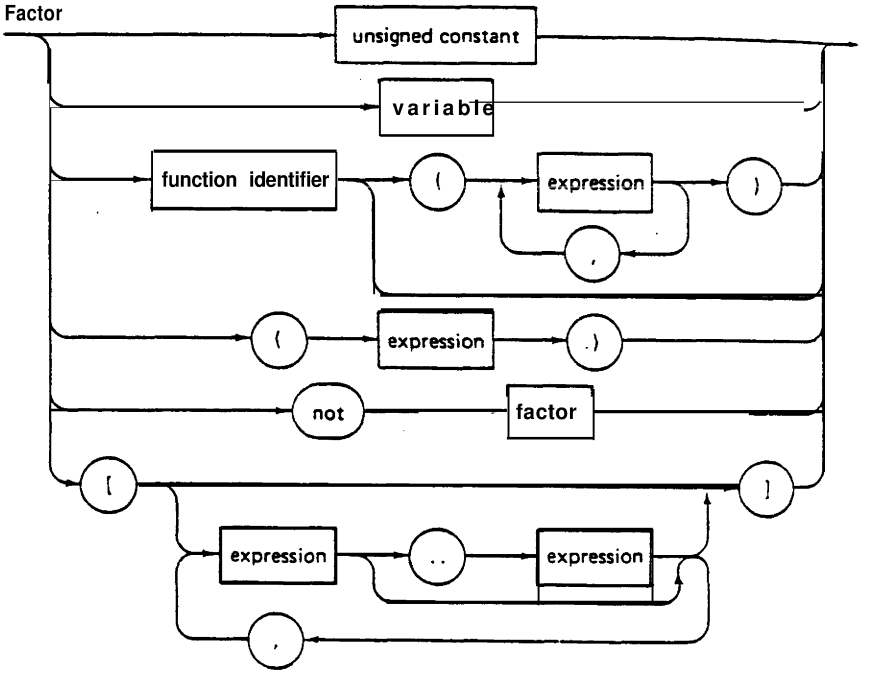
(a)[5 1 Prove that the firehouse problem is IV-complete.

(b) [5] Suppose d is fixed instead of being an input parameter. Is the firehouse problem still NP-complete?

(c)[5 1 Suppose k is fixed (but d is variable). Is the firehouse problem in this version NP-complete?

1. Syntax Notation (10)

a.(5) Write the syntax definition for FACTOR, given below in PASCAL diagram form, using BNF.



b.(5) Discuss briefly the advantages of the two notations.

2. Paging (10)

a. (3) Specify the information needed by a paging system to decide when to release or rewrite memory pages to the paging device.

b. (3) Why are larger pages better than small ones?

c. (4) List four parameters that affect optimal page size and give their effect.

3. Language (11)

A PASCAL record can have variant parts.

a. (1) Why are these variants desirable?

b. (1) What is the problem with them?

c. (2) Suggest a solution to overcome the problem.

d. (3) You are writing these new PASCAL records out to a file, another program is to read them. What happened to your solution?

e. (4) How would you fix that?

4. Ethernet Communication Protocol (5)

a. (1) In this protocol, who does error detection and correction?

b.. (4) How is that done? Write steps in a very high level language format.

5. Code Generation (6)

Write in an assembly language [either HP2xxx, DEC10, IBM370, DEC11, or MIX] the code to be generated for the CASE statement in the PASCAL program given below.

If no match is obtained the case statement should be skipped.

(Do not concern yourself with syntactical details of the language.)

```
PROGRAM  SALARYLIMITS
VAR ED, SAL, BSSAL, MSSAL, OTHERSAL : INTEGER;
    BEGIN
        CASE ED OF
            1 : SAL := PHDSAL;
            2 : SAL := MSSAL;
            3 : SAL := BSSAL;
            9 : SAL := OTHERSAL
        END;
    END.
```

6. Cooperating Processors (10)

a. (5) You have multiple processors sharing memory. A processor can lock units of memory.

What are two protocols to prevent deadlock?

b. (5) Which protocol would be best in a distributed computing environment, i.e., the processors communicate to storage via potentially slow communication lines, and why?

7. Binding (8)

Describe the difference in argument binding between ALGOL and LISP. Show an example.

Problem 1 (a),(b)

There are several possible approaches to this problem. For instance, it can be solved by sorting the line segments either by x or by y co-ordinates. We shall describe a solution that sorts by x co-ordinates that is easily adaptable to part (c).

1. Sort the endpoints of the intervals (both left and right) by x co-ordinate.

2. Scan through the endpoints in increasing order on x-co-ordinate. Maintain a balanced tree of line segments having a point with the current x-co-ordinate. These segments are ordered in the tree by y-co-ordinate.

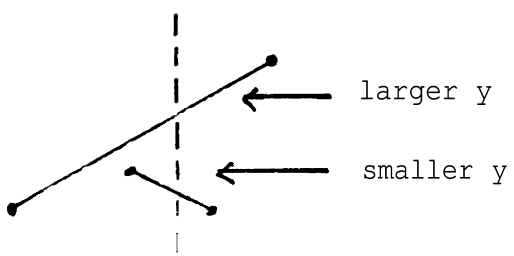
(i) To process a left endpoint a_i : Enter i with value y_i in the tree. If i is the segment in the tree with smallest y-value, mark i visible.

(ii) To process a right endpoint b_i : Delete i from the tree. Mark visible the segment in the tree with smallest y-value.

After processing all endpoints, the ones marked visible are indeed visible; the others are not. Same care must be taken to deal with ties in the x-co-ordinates of endpoints, depending upon how exactly one wishes to define "visible".

This algorithm requires O(n log n) time for the sorting of 2n numbers in (i) and O(n log n) time for n insertions and n deletions in (ii); each tree operation takes O(log n) time using your favorite balanced tree data structure.

(c) The crucial point is that if two intervals overlap, then anywhere along their overlapping part the order of their y co-ordinates stays fixed, because they don't intersect:



We use the same tree and operations as in (a) and (b), but during every insertion and deletion we recompute y co-ordinates of segments along the search path. Each such recomputation takes constant time, and the modified algorithm has an O(n log n) time bound.

Problem 2 (a),(b).

Define $s(i,j)$ for $0 \le i \le n$, $0 \le j \le n$ to be true if $c_1, c_2, \ldots, c_{i+j}$ is a composition of $a_1, a_2, \ldots$ and $b_1, b_2, \ldots, b_j$, and false otherwise. We want the value of $s(n,n)$. We can compute the values $s(i,j)$ by using the recurrence

$$s(0,0) = \text{true}$$

$$s(i,j) = (s(i,j-1) \text{ and } c_{i+j} = b_j) \text{ or } (s(i-1,j) \text{ and } c_{i+j} = a_i).$$

A double loop iterating over i and j in increasing order computes $s(i,j)$ in $O(n^2)$ time.

Problem 3.

Divide the sheds into $\lceil \sqrt{n} \rceil$ (or fewer) groups, each containing $\lceil \sqrt{n} \rceil$ bikes. We maintain the property that if any bikes are in a group, they are packed into the first sheds of the group. We also make sure that, although bikes are moved, no bike moves from one group to another.

On arriving: Check the last shed of every group to find a group with an empty shed. Check all sheds in the group to find the first empty one. Put your bike in it. Time: $O(\sqrt{n})$.

On leaving: Check all sheds in the group in which you left your bike to find your bike. Remove your bike and replace it by the bike in the last filled shed in the group. (If your bike was the last, no replacement is necessary.) Time: $O(\sqrt{n})$.

Problem 1.

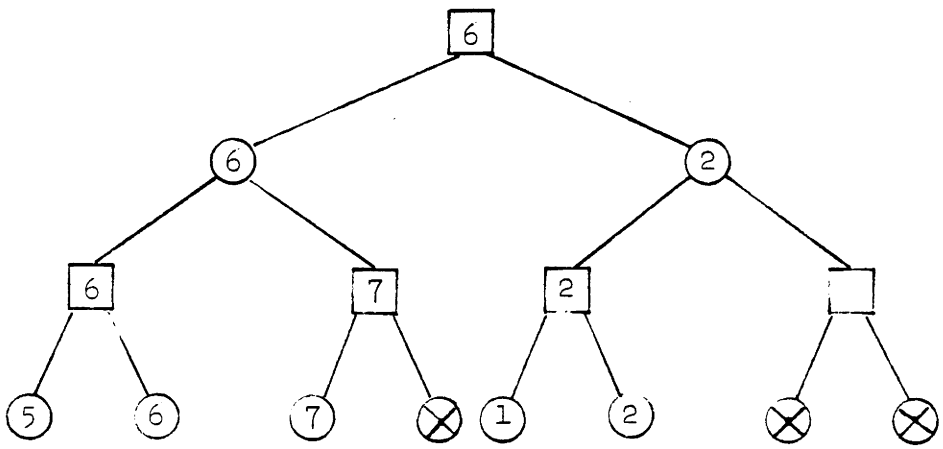
(a) A semantic network saves space when the data base is sparse, i.e., not every relation has a value for every entity.

(b) **Backward chaining** is a problem solving method in which one chooses an operation that achieves a goal, then sets up the prerequisites for that method as subgoals. 'There is no constraint on how the operations are chosen.

GPS uses a table of differences to determine which operation to perform.

(c) **Meta-planning** is planning applied to the planning process itself. The following are two typical heuristics.

1. Solve hard problems before easy ones.
2. If you can't prove a claim, try to disprove it.



Problem 2.

(a) 1. $\exists h$ Hat(h) A Belongs(h, The-Postman) $\wedge$ COLOR(h, BROWN) $\wedge$ LOC(h, Cleaner)

2. $\forall x \exists y$ Loves(x,y)

$\exists y \forall x$ Loves(x,y)

3. $\exists p_1 \ldots p_{23}$ Postman(p_1) A ... $\wedge$ Postman(p_{23})

A $p_1 \neq p_2 \wedge p_1 \neq p_3 \wedge .. 0 \wedge p1 \neq p_{23}$

A $p_2 \neq p_3 \wedge \ldots$

(b) 1. u/f(A) , v/A , w/A , x/A , Y/A

2. none

Problem 3.

(a) He should maximize points/ expected time. This isn't optimal because when he comes near the end, he may choose a problem that requires more time than he has left. In other words, he must take into account the amount of time he has remaining.

(b) Branch and Bound is most appropriate with time as her cost function and 20 points on her goal.

(c) He is correct; Beatrice's strategy is not optimal for him. He should maximize points/time subject to the constraint that he gets 20 points in each area. In minimizing time, Beatrice might choose a problem for which points/time is not high.

(d) The robot would have to know the goal of the test and the characteristics of each search method.

Problem 4.

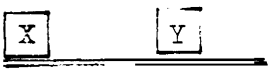
(a) prerequisites for PUTON(x,y): CLEAR(x)
CLEAR(y)
$x \in$ BLOCK
$y \in$ BRICK $\cup$ TABLE

delete list: ON(x,z)
CLEAR(y)

add list: Clear(z)
ON(x,y)

(b) PUTON(P1, TABLE)
PAINT(B1, Red)
PUTON(B1, B2)

(c) The following sequence of operations would teach Winston's program the -prerequisites for PUTON. In each case, assume that the goal is to put X into Y .

1. 

2. 

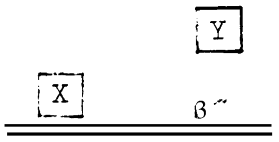
generalizes X to manipulable objects

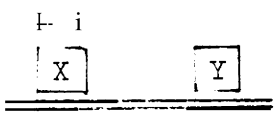
3. 

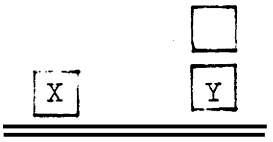
requires Y to be a block

4. 

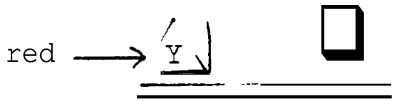
demonstrates that X need not be on the table

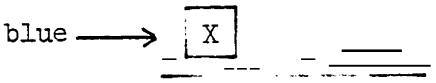
5.  demonstrates that Y need not be on the table

6.  requires that X be clear

7.  requires that Y be clear

(d) If 2 differences are introduced at once, the program might concentrate on the wrong property. For example, suppose examples 1 and 2 were replaced with the following.

1. 

2. 

The program might infer that a block must be red to be moved and might not realize the restriction on Y .

(e) One could either request it to put a block into a pyramid or to move a blue block. In the first case the program wouldn't realize that the situation was impossible; in the second it would spuriously paint the block before moving it.

1. Hardware

a) Needed at CALL:

i. Register to save current instruction counter.
ii. Current instruction counter gets new address.
iii. Parameter values or address must be obtainable (perhaps via i or ii).
iv. General purpose registers have to be saved or protected.

b) i. Single word parameters are most efficient using value, or if needed, value-result.
ii. Large parameters are best passed by reference, to avoid large moves.

c) To handle recursion a stack is needed to handle previous past instruction counters (a.i) and parameters (b.i). Arguments, otherwise passed by reference, may have to be stacked if arbitrary computation is possible at intermediate levels.

2. High Speed Arithmetic

In a pipelining machine information about instruction which may cause a floating point exception has to be carried along until no further failures are possible. Enough information has to be kept of all instructions to allow restart - this means mainly that no stores precede earlier floating point stores.

Simply flushing the stack on detection of floating point error leads to problems in error indication to the user and in programming of adequate error recovery.

3. bit 1: sign bit; if $\begin{matrix}0\\1\end{matrix}$ number is $\begin{matrix}\text{nonnegative.}\\\text{negative}\end{matrix}$

bits 2-6; exponent + 16; thus the possible binary values 00000 through 11111 = 31_{10} represents exponents -16_{10} through +15. The base is 4.

bits 7-18: mantissa, a 6 digit base 4 quantity with point between the first and second quaternary digits. If bit 1 is 1 the mantissa holds the 1's complement of the magnitude.

Numbers which cannot be represented exactly (e.g., 1/3, 2/3) are chopped.

4. Communication Alternatives

In Synchronous Communication transmission is continuous. When no data are available, IDLE characters are put into the transmission stream. Clocks are derived from the signal.' Transmission can proceed at a high rate because of the stability of transmission.

In Asynchronous Communication the transmission is initiated when data are available. A start bit precedes the data, and at least one stop bit is inserted to account for differences in clock initiation time and clock speed, prior to transmitting more data. More bits are transmitted per data element and transmission rates tend to be lower.

5. If $x_4x_3x_2x_1$ represent a 4 digit binary number between 0 and 15, then $f(x_4,x_3,x_2,x_1)$ has the given table of values, f is true if $x_4x_3x_2x_1 \in \{0001, 0010, 0011, 0101, 1000, 1101\}$

x_1x_2 \ x_3x_4	00	01	11	10
00	0	0	0	1
01	1	1	1	0
11	1	0	0	0
10	1	0	0	0

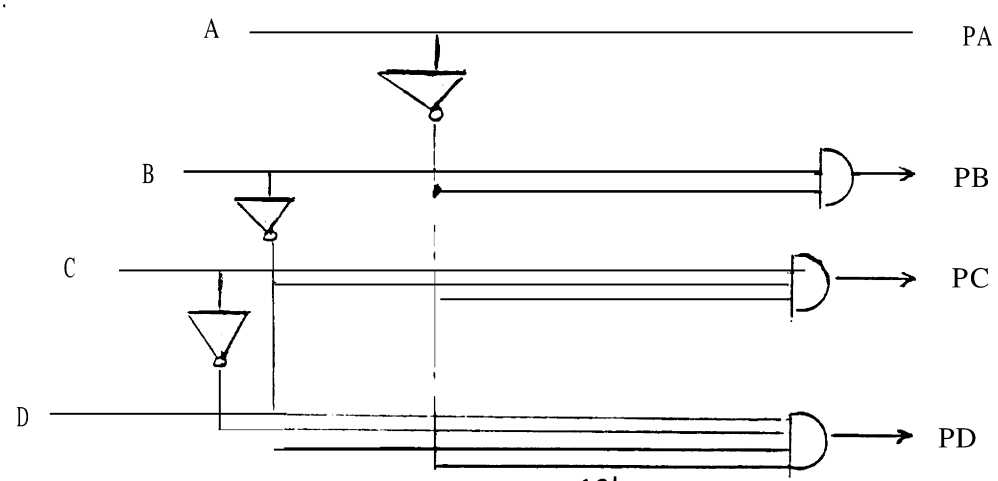
A minimal sum of products form is $f = \bar{x}_3\bar{x}_4x_1x_2 + x_4\bar{x}_1x_2 + x_3\bar{x}_1x_x + x_1\bar{x}_3\bar{x}_4$

Since a minimal sum of products for $f' = x_1x_4 + x_1\bar{x}_3 + x_2\bar{x}_4 + x_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3$

Then

$$f = (\bar{x}_1 + \bar{x}_4)(\bar{x}_1 + \bar{x}_3)(x_2 + \bar{x}_4)(\bar{x}_2 + \bar{x}_3 + x_4)(x_1 + x_2 + x_3)$$

6.



1. a) $\det(A) = \prod_{i=1}^{n} u_{ii}$

   b) $n^3/3$

   c) $> n!$

2. a) Observe that the factor L can fit in the nonzero part of A :

   $\ell_{ij} = 0$ if $j < f_i$, $1 \leq i \leq n$.

   Modify the loop bounds in Cholesky algorithm to avoid operating on zeros:

```
for r:= 1 to n
    for c:= f_r to r-1
        ℓ_rc := a_rc ;
        for k:= max(f_r,f_c) to c
            ℓ_rc := ℓ_rc - ℓ_rk * ℓ_ck ;
        ℓ_rc := ℓ_rc/ℓ_cc ;
    ℓ_rr := a_rr ;
    for k := f_r to r-1
        ℓ_rr := ℓ_rr - ℓ_rk * ℓ_rk ;
    ℓ_rr := sqrt(ℓ_rr) ;
```

   b) Define $w_j \equiv \text{card}\{i>j \mid f_i \leq j\}$

   for $j = 1,\ldots,n-1$.

   Ex: A has the structure

```
X
x x
  x x
x x x x
  x x x x
```

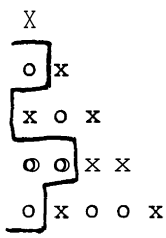
So that $f_1 = 1,\ f_2 = 1,\ f_3 = 2,\ f_4 = 1,\ f_5 = 2$.

Then $w_1 = 2,\ w_2 = 3,\ w_3 = 2,\ w_4 = 1$.

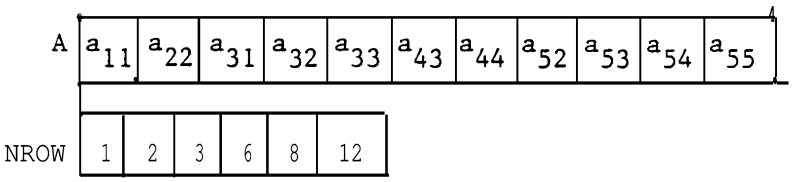
The # of mults $= \sum_{j=1}^{n-1} w_j + \frac{w_j^2}{2}$

2. **c)** It is reasonable to store only the elements a_{ij} with $f_i \leq j \leq i$ and use the same space for L (i.e. overwrite A with **L).** These would be stored in a 1-dimensional array A , row by row, in order of increasing column index within the row. An array of n + 1 pointers NROW would be used to show where the rows begin.

Ex: The lower triangle of A :



would be stored



NROW(i) = location in A of first element of row i

NROW(m+1) = location of first unused position in A .

3. Subroutine for $\sqrt[3]{c}$.

I. Express c as $c = \hat{c} \times 2^{3n}$

where $1 \leq |\hat{c}| < 8$. Then $\sqrt[3]{c} = \sqrt[3]{\hat{c}} \times 2^{n}$.

This can be done on a binary computer without introducing roundoff.

II. Use Newton's method to solve the equation $f(x) = x^3 - \hat{c} = 0$.

This means, given x_0,

$$x_{n+1} = x_n - \frac{x_n^3 - \hat{c}}{3x_n^2}$$

$$= \frac{1}{3}\{2x_n + \frac{\hat{c}}{x_n^2}\}$$

III. Since $|\hat{c}| \geq 1$ and $|\hat{c}| < 8$,

$$1 \leq |\sqrt[3]{\hat{c}}| < 2 .$$

Take some rough approximation to $\sqrt[3]{\hat{c}}$, by a polynomial or piecewise polynomial, to obtain the initial approximation x_0 .

For example, take

$x_0 = 1.5$ with rel. error at most .5

or $x_0 = \begin{cases} 1.25 & \text{if } \hat{c} \leq (1.5)^3 \\ 1.75 & \text{if } \hat{c} > (1.5)^3 \end{cases}$ with rel. error $\leq$ .25

or $x_0 = 1.5 + \left(\frac{2\hat{c} - 9}{14}\right)$ with rel. error $\leq$ .162

IV. It was difficult to carefully estimate the number of iterations required. With

$$e_n \equiv x_n - \sqrt[3]{\hat{c}}$$

and the assumption $|e_0| \leq .25$, it can be shown that $|e_5| \leq 2^{-33}$ (in the absence of round-off error), so 5 iterations would suffice.

In fact, since

$$e_{n+1} \leq e_n \frac{f''(\xi)}{2f'(x_n)} \quad n$$

where $\xi \in$ int $(x_n, \sqrt[3]{\hat{c}})$, it follows that

$$|e_{n+1}| \leq 2|e_n|^2, \quad |2e_{n+1}| \leq |2e_n|^2 .$$

This is because the iterates x_n remains in the interval

[1,10/3 and in that

$$\left|\frac{f''(\xi)}{2f'(x_n)}\right| \leq 2.$$

Thus,

$$2|e_n| \leq |2e_{n-1}|^2 \leq |2e_{n-2}|^4 \leq \ldots |2e_0|^{2^n}$$

$$|e_n| \leq \frac{1}{2}|2e_0|^{2^n} \leq 2^{-(1+2^n)}$$

whence $e_5 \leq 2^{-33}$.

Since the iteration function does not require subtracting nearly equal quantities, round-off should be no problem. If full 24 bit accuracy was required, however, the last Newton step would be done in double precision.

PROGRAM

```
function cuberoot (c);
   real cuberoot, c,x,cbar;
   integer iexp;
/* find cbar in[1,8] such that */
/* c = cbar * 8↑iexp

   cbar:=c;
   iexp:=0;
   while (cbar ≤ 1) do
         {cbar:=cbar * 8;
          ixep:=iexp - 1}
   while (cbar ≥ 8) do
         {cbar:= cbar /8;
          iexp:= iexp + 1}
/* generate initial guess. error ≤ .25 */
   if (cbar ≤ 27./8.) then x:= 5/4
                      else  x:= 7/4;
   x:= 1/3 * (2*x + cbar/(x*x));
```

repeat 4 more times { }

```
   cuberoot := x * 2↑iexp
   end(cuberoot);
```

4. a) Since $r'(x) = 3x^2 - 3/4$, local extrema may occur at $-1,1$ (the endpoints) and at $-1/2,1/2$ (the critical points). One simply computes $r(-1) = r(+1/2) = -1/4$, $r(-1/2) = r(1) = \frac{1}{4}$. Thus $\|r\| = 1/4$.

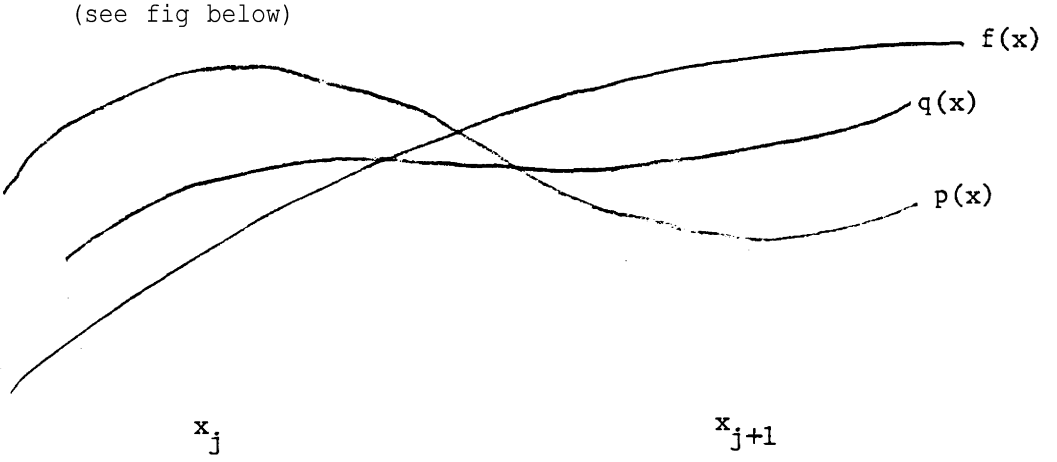
b) r attains its maximum with alternating sign at 4 points in $[-1,1]$. Thus the hypotheses of the theorem are fulfilled with $n = 2$, $f(x) = x^3$, $p(x) = \frac{3}{4}x$.

c) Suppose $\|f-q\| < \|f-P\|$, where p,q are polynomials of degree n , $f \in ([-1,1]$, and $f-p$ achieves its max with alternating sign at distinct points $x_1, x_2, \ldots, x_{n+2} \in [-1,1]$. Since

$|(f-q)(x_j)| \leq \|f-q\| < \|f-p\| = |(f-p)(x_j)|$,

$q(x_j) < p(x_j)$ when $p(x_j) > f(x_j)$, and $q(x_j) > p(x_j)$ when $p(x_j) < f(x_j)$

(see fig below)



Thus $p-q$ alternates in sign at the $n+2$ points $\{x_j\}$, and therefore has $n+1$ roots in $(-1,1)$. Thus $p-q \not\equiv 0$, which contradicts the hypothesis.

1, Yes. Let xi be the ith bit of x, yi the ithe bit of y, and zi the ith bit of $z=x+y$. We wish to show that z is recursive if both x and y are recursive. Now, for each i, z_i can be computed from Xi and yi if it is known whether there is a carry from lower order bits. This can be decided by searching down the strings x and y looking for a pair of zeroes (no carry), or a pair of ones (carry). If the search terminates, then fine. Otherwise z is recursive: it is some finite bit string followed by an infinite string of ones.

More formally, for each i there are two cases to consider.

(1) $x_j \neq y_j$ for all $j > i$.

(2) There is some $k>i$ such that $x_k = y_k$.

If there is any i such that case (1) holds, then z is recursive, since zj = 1 for all j>i. So we may assume that for each i there is some k>i such that $x_k = yk$. But then each z_i can be effectively computed by searching for the first k>i with $x_k = y_k$; if $x_k = yk = 0$ then there is no carry; if $x_k = y_k = 1$ then there is.

Note that although x + y is recursive for any recursive x and recursive y, it is not possible to pass recursively from indices of Turing machines for x and y to the index of a Turing machine for x + y. This is because it is impossibe to tell in general which of the cases (1),(2) above holds.

2.. For each n, let Ψ_n be a sentence which says, "there exist at least n distinct objects". Ψ_n might be, for example, "$\exists x_1 \exists x_2 \ldots \exists x_n (\wedge_{1 \leq i < j \leq n} (x_i \neq x_j))$". Then by the hypotheses of the problem, every finite subset of the set of sentences $S = \{\varphi, \Psi_1, \Psi_2, \Psi_3, \ldots\}$ has a model. Therefore, by the compactness theorem, the whole set S has a model: that model will be a model of φ, and will have more than n elements for each n; that is, it will be an infinite model.

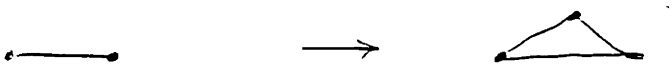
3. Upper bound = m^n. Let T be any derivation tree for the sentence S. Then no non-terminal symbol of the grammar can appear twice along one path down the tree. If a non-terminal did appear twice on one branch, the subtree rooted at the upper appearance (assuming the root of T is at the top) could be replaced by the subtree rooted at the lower appearance, yielding a derivation of a different sentence, contrary to hypothesis. Since the grammar has n productions, only at most n distinct non-terminals may appear along any path down the tree. Thus, the depth of T is at most n. The branching factor is at most m, and so the maximum possible number of leaves which T can have (= the maximum possible length of S) is m^n. This upper bound is achieved by the grammar:

$S \rightarrow A_2A_2 \ . \ . \ .A,$
$A_2 \rightarrow A_3A_3 \ . \ . \ .A,$
$A_3 \rightarrow A_4A_4 \ . \ . \ .A,$
. . .
$A_n \leftarrow aa \ . \ . \ . \ a$

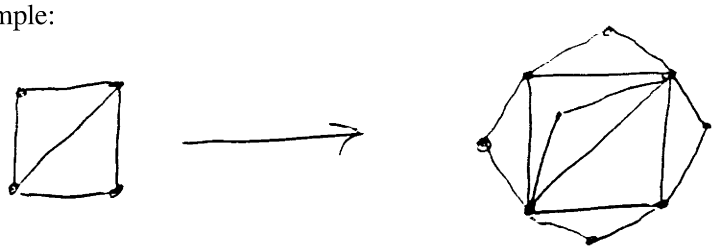
where the length of the right hand side of each production is exactly m.

4. (a) The firehouse problem can be solved in NP time by first guessing a placement of the firehouses and then checking whether that placement has the desired properties (it is easily seen that the check can be done in deterministic polynomial time). The problem is NP hard because the vertex-cover problem can be reduced to it, as follows. (Also, the more obscure *dominating set problem* [Garey & Johnson pg 190] reduces directly to the firehouse problem by taking d= 1)

Suppose that we wish to know whether a gragh G can be vertex-covered by k vertices. Take G, and make each of its edges into a triangle by adding a new vertex and two new edges:



For example:



Let G' bc the graph which results from G by this procedure. I claim that G has a vertex-covering with k firchouses if and only if G' has a firehouse covering with k firehouses and with d= 1. The implication in one direction is trivial: a vertex covering for G is a firehouse covering for G'. For the converse, suppose that we have a firchousc-covering of G' with k firchouses. Consider the set of new vertices which were added to G in order to arrive at G'. If a firehouse sits on any of the new vertices, then it can be moved to either of the neighboring old verices without destroying the covering (if either of the neighboring vertices already has a firehouse, the firehouse on the new vertex can be simply removed). Thus if G' is firchousc coverable at all, it is firchouse coverable subject to the restriction that all firchouses sit on old vertices. It is easy to see that a firehouse cover of G' with d= 1 and all firchouses on old vertices is a vertex cover of G, and so wc are done.

(Note that although a vertex cover is always a firehouse cover with d=l, the converse is not true. For example, the triangle requires only one firehouse, but two vertices are needed for a vertex cover.)

(b) Yes, if d is fixed with d=1.

(c) No. For each k, there are less than n^k possible placements of firehouses on a graph with n vertices. Thus all possibilites for a firehouse cover of G can be checked in time polynomial in the size of G.

SOLUTIONS

SOFTWARE SYSTEMS - May 1980

1. Syntax Notation

a)
```
<factor> ::= <unsigned constant>|
             <variable>|
             <function>
             (<expression>)|
             NOT <factor> |
             [set list]
<function> ::= function identifier
               function identifier (expression list)
<expression list> ::= <expression>|      {note direction of recursion
                      <expression list>,<expression>
<set list> ::= <null>|
               <set specification>|
               <set specification>,<set list>
<set specification> ::= <expression>|
                        <expression>..<expression>
```

b) The syntax diagrams are easy to follow when writing statements.
Multiple recursions can be incorporated in one definition chart.
BNF can be analyzed to deduce syntax features of the grammar.
BNF can be processed automatically to generate parsers.
BNF makes recursion clear.
Diagrams can be generated from BNF.

2. Paging

a) Basic paging system needs
used, changed, shared, locked-for-I/O indicators for a page.
If working set algorithm is used, also needs owner and owner status.
IF LRU is used, time or relative time since last use.
The changed or dirty bit determines release versus rewrite.

b) Less bookkeeping and smaller pagetables,
less overhead, less paging in well-behaved programs.

c) Disk or drum seek latency and seek
If fast, smaller page size is feasible.
Average user program size relative to real memory
Number of users relative to real memory
If many users, smaller page size is warranted.
Disk or drum transfer rate
If high, larger pages may be read in.
Expected user program behaviour
Segment sizes in user program
If very random accesses then segments are small, and smaller page size is best
Match of device sector and page size is desirable
Page table limits and cost
If page table are restricted and costly, large pages are best.

3. Language

a) Variants are used to describe entities that have alternative attribute types,
Reduces storage

b) Variants cannot be type-checked properly.

c) Enforce the use of tags to indicate which variant is in use, perhaps with case constructs on reading records. [See Algol 68]

d) Types and tags depend on programs, and another reading program may define record structure differently,

e) Copy record description to file, and check with programs that read file, or actually import the record description.

4. Ethernet

a) The sender detects collisions and retransmits.

b)
```
PROCEDURE TRANSMIT (MESSAGE, LENGTH-OF-MESSAGE);
    MULTIPLIER = 1
R1:   READ ETHER INTO BUSY;
    IF BUSY THEN GOTO R1;
    FOR I = 1 TO LENGTH OF MESSAGE BEGIN:
          BIT = MESSAGE(T);-
          WRITE AND READ BIT ON ETHER;
          {WRITE AND READ SEPARATED BY NET TRANSMIT TIME1
          IF BIT ≠ MESSAGE (I)
               THEN GOTO TRYAGAIN;
    END;
    RETURN;
TRYAGAIN:  WRITE JAM ON ETHER {ASSURE FAILURE IS DETECTED BY ALL}
          DELAY (RANDOM*MULTIPLIER)
          MULTIPLIER = MULTIPLIER"2
    GO TO R1;

END.
```

5. Code Generation

```
* CODE FOR GENERAL MACHINE WITH REGISTERS R1, R2, R3
* Indexes, specified as (R) are added to address
* TABLE SOLUTION

      LD    A1, ED
      CMP   A1, '1'
      JLS   DONT
      CMP   A1, 'g'
      JGT   CONT
      LD    A2, TAB-1(A1)
      LD    A3, 0(A2)
      STO   A3, SAL
DONT . . .
      ...
TAB   ADR   PHDSAL    1
      ADR   MSSAL     2
      ADR   BSSAL     3
      ADR   SAL       4
      ADR   SAL       5
      ADR   SAL       6
      ADR   SAL       7
      ADR   SAL       8
      ADR   OTHERSAL  9
```

(8 executed instructions)

```
* FIXED OPT BRANCH SOL'N

      LD    A1, ED
      CMP   A1, '3' MIDDLE
      JLS   LOW
      JGT   HIGH
      LD    A2, BSSAL
      JMP   DONE
LOW   CMP   A1, '1'
      DLS   DONT
      JGT   IS2

      LD    A2, PHDSAL
      JMP   DONE
IS2   LD    A2, MSSAL
      JMP   DONE
HIGH  CMP   A1, 'g'
      INE   DONT
      LD    A2, OTHERSAL
DONE  STD   A2, SAL
DONT . . .
```

(7-9 executed instructions)

```
* DEC-10    MINSIZE
MOVE R1, ED
MOVE R2, SAL
CAIN R1, 1
MOVE R2, PHDSAL
CAIN R1, 2
MOVE R2, MSSAL
CAIN R1, 3
MOVE R2, BSSAL
CAIN R1, g
MOVE R2, OTHERSAL
MOVEM R2, SAL

(11 exec. instr.)

[CAIN = COMPARE AND
 SKIP IF REG. NEQ
 IMMEDIATE OPERAND]
```

6. Cooperating Processes

a) i. Uninterruptible initial locking of all resources to be used.

ii. Acquire resources using a globally specified ordering.

iii. Do not assign initial or additional resources if not sufficient resources are available for process completion (Bankers algorithm).

iv. Allow pre-emption of processors.

b) i. Assigns resource with little communication. During acquisition all other processors are locked out.

ii. Ordering can be made known to all processors, decision making is distributed.

iii. Is not very suitable, implies central controlling node, although potentially best storage allocation.

iv. Remote' processors are not well enough controlled to use preemption.

7) Binding

While variable binding is static in ALGOl and dynamic in LISP, arguement binding on call by NAME is dynamic, depending on callers environment at time of use. LISP NLAMBDA function arguments are evaluated at time of use, using current environment, unless a FUNARG is used in the invocation, referring to a specific point in the evaluation stack. LISP LAMBDA is similar to ALGOL CALL-BY-VALUE

ALGOL

```
PROC P(A, B, C);
    B= 5;
    PRINT(A, B, C)
END;
    A = 2
  B=4
    CALL P(A, B, A+B)
END.
YIELDS
2,5,7
```

LISP

```
(DEFUN K (A B C)
   (PROG (SETQ B 5)
         (PRINT A B C))
(PROG
(SETQ A 2)
(SETQ B 4)
(K A B (PLUS A B))
     )
YIELDS
2 5 6
```

Spring 1980 - Comprehensive Programming Project

Approximation of Integrals

The goal of this project is to construct a program which will compute approximations of integrals of real valued functions over convex regions of $\mathbb{R}^2$ which are bounded by polygons.

This programming problem is associated with numerical analysis but it is our intention that the numerical analysis aspects of this project are sufficiently well described here for the project to be completed satisfactorily. More sophisticated ideas could be used for error estimation, etc., but we have intentionally kept these as simple as possible. Extended precision arithmetic will not be necessary. If you have any questions related to the numerical analysis aspects of the problem, feel free to ask those members of the exam committee listed at the end of this project description for clarification.

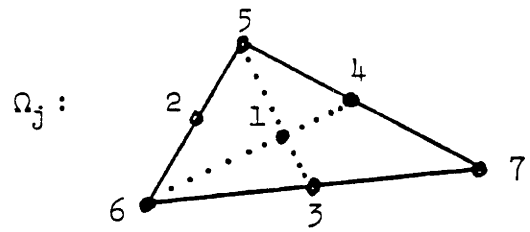
The programs will be evaluated based upon the efficiency of the algorithms constructed and the underlying data structures, the structure and clarity of the code, and the associated documentation. You will be asked to develop a heuristic strategy to efficiently implement the basic algorithm to be described. You should describe the heuristic you use and explain why you find it to be a reasonable approach to the problem.

The program to be developed will be an adaptive quadrature program based upon a quadrature formula defined on triangles. We assume that we are developing programs for approximating the integrals of functions which are very expensive to evaluate. It is our goal to construct programs which attempt to minimize the number of function evaluations of the integrand. In line with this goal we require that the programs never evaluate the integrand at any point of $\mathbb{R}^2$ more than once. This also makes the programming more interesting.

The Quadrature Formula and Error Estimates,

Our basic approach to the approximation of an integral will be to subdivide the polygonal region into triangular subregions, approximate the integral over each subregion using a quadrature formula, and finally to add up these approximations. The program should generate a sequence of finer and finer triangulations, based upon error estimates, until it obtains one for which the total-estimated error is sufficiently small.

If we label the vertices, midpoints of the sides, and the centroid of a triangle Ω_j as



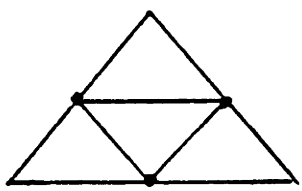
we define our basic quadrature formula which approximates $\int_{\Omega_j} f$ as

$$(1) \qquad I_{h,\Omega_j}(f) = \mu(\Omega_j) \sum_{\ell=1}^{7} w_\ell f_\ell$$

where f_ℓ is the value of the integrand in the point ℓ and $w_1 = 27/60$, $w_2 = w_3 = w_4 = 8/60$ and $w_5 = w_6 = w_7 = 3/60$. $\mu(\Omega_j)$ is the area of the triangle. This formula is exact for cubic polynomials and has an error which is $O(h^4)$ as $h \to 0$ where h is the maximum of the lengths of the sides of the triangle, i.e.,

$$\int_{\Omega_j} f - I_{h,\Omega_j}(f) = O(h^4) ,$$

provided that the integrand, f, is sufficiently smooth. In order to estimate errors, and to improve our approximations, we will also consider approximations on Ω_j obtained by subdividing Ω_j into four similar triangles such that the new vertices are the midpoints of the sides of the original triangle.



We label these $\Omega_{j,\ell}$, $\ell = 1,2,3,4$. We can then approximate the integral of f over $\Omega_{j,\ell}$ using (1) and over Ω_j by their sun, i.e., by

$$I_{h/2,\Omega_j}(f) = \sum_{\ell=1}^{4} I_{h,\Omega_{j,\ell}}(f) .$$

We can then estimate the absolute error

$$E_{h,\Omega_j} = \left| \int_{\Omega_j} f - I_{h,\Omega_j}(f) \right|$$

by

$$(2) \qquad e_{h,\Omega_j} = \left| I_{h/2,\Omega_j}(f) - I_{h,\Omega_j}(f) \right| .$$

If our error estimate (2) is not good enough we then take the four subtriangles just defined and subdivide each of them to estimate the error on each of them, etc. This will be our basic refinement procedure.

For any given integration (there will be several) you will be given the vertices of the boundary polygon, a definition of the function $f(x,y)$ to be integrated, and an error tolerance ε . You are asked to input an initial triangulation of the region and the error tolerance. You should write a procedure to compute values of f . This initial triangulation should contain as few triangles as possible consistent with the desire that the ratios of shortest to longest sides should be close to 1 -- this is to be interpreted rather loosely, i.e., in spirit.

Something Else To Do.

We ask that you consider the problem of constructing a program to compute a good initial triangulation. A good initial triangulation will be as above -- the ratios of shortest to longest sides should be nearly 1 if possible, Discuss approaches and difficulties associated with this problem and describe how you would implement such a program in your writeup. Do not program this part of the problem.

Refinement Strategies.

The crucial part of this project is to devise a strategy to decide when and where to refine the triangulation. We now discuss several issues which you should consider in devising your strategy.

Let Ω be the region over which we are to approximate the integral of a given function f and let Ω_j , $j = 1,2,\ldots,m$ be an initial triangulation of Ω , i.e., $\Omega = \bigcup_{j=1}^{m} \Omega_j$. We begin by considering the basic strategy of eqtidistributing the absolute error over the Ω_j . We will modify this by changing our strategy twice before we complete the description of this programming problem. We want to construct an approximation $I_\Omega(f)$ such that

$$\left| \int_\Omega f - I_\Omega(f) \right| \leq \varepsilon \ .$$

We define the error per unit area, ε' , by

$$\varepsilon' = \varepsilon / \mu(\Omega)$$

where $\mu(\Omega)$ is the area of the region Ω . if we generate a triangulation $\Omega = \bigcup_j \Omega_j$ such that

(3) $$\left|\int_{\Omega_j} f - I_{h,\Omega_j}(f)\right| \leq \varepsilon' \mu(\Omega_j)$$

we obtain

$$\left|\int_{\Omega} f - I_{\Omega}(f)\right| \leq \varepsilon$$

with

(4) $$I_{\Omega}(f) = \sum_j I_{h,\Omega_j}(f) \quad .$$

We will estimate ε' in (3) using (2). Let $e'_{h,\Omega_j} = e_{h,\Omega_j}/\mu(\Omega_j)$ be our estimate of ε'. Note that we must refine Ω_j to compute our error estimate. Since we must compute $I_{h/2,\Omega_j}(f)$ to estimate the error we might as well use Richardson extrapolation to try to improve our estimate of the integral on Ω_j which we finally use. We will use

$$I^*_{h,\Omega_j} = (16 I_{h/2,\Omega_j} - I_h)/15$$

as our final approximation on Ω_j if (3) holds and substitute these quantities for $I_{h,\Omega_j}(f)$ in (4).

We now discuss the first of our improvements on this strategy. We may have done much better than $\varepsilon'\mu(\Omega_j)$ on some of the regions Ω_j, i.e., $e'_{h,\Omega_j} \ll \varepsilon'$ for some j. We can take advantage of this by relaxing our error tolerance on the remaining regions. Suppose at the n-th step in our algorithm we have accepted approximations on the n subregions Ω_j, j = 1, . . .,n, and at this step we have estimates of the errors per unit area e'_j. We now define $\varepsilon'_{(n+1)}$ by

$$\varepsilon'_{(n+1)} = \left(\varepsilon - \sum_{j=1}^{n} e'_j \mu(\Omega_j)\right) \Big/ \mu\left(\Omega - \bigcup_{j=1}^{n} \Omega_j\right) \quad .$$

We now use this for ε' in (3) for j = n+1. To begin we define $\varepsilon'_{(1)} = \varepsilon'$.

Finally, we observe that this strategy could be modified to advantage for many common situations. We assume that our integrands are only difficult to approximate over isolated and small subregions of the region Ω. We may encounter a subregion over which the integral is very difficult to approximate -- we are required to make repeated subdivisions. If the integral is easily approximated over the rest of the region we may do much better (in terms of function evaluations) if we require more accuracy on these "easy" subregions and less on the "difficult" subregions.

Accordingly, you can provisionally accept approximations over subregions but be prepared to go back if you encounter a difficult subregion later, Develop an heuristic strategy to use in your program which takes advantage of this.

Output.

For each problem we ask that you output:

(I) the number of initial triangles,
(2) a description of the initial triangulation,
(3) the number of times a triangle is subdivided,
(4) the number of times the integrand is evaluated,
(5) your approximation of the integral,
(6) your estimate of the absolute error.

Problem 1.

Let Ω have the vertices

$$(0,0)\ ,\ (1,0)\ ,\ (2,1)\ ,(1,2)\ ,\ (0,2)\ ;$$

f defined by

$$f(x,y) = x^2 + xy + y$$

and

$$\varepsilon = 10^{-4}\ .$$

Problem 2.

Let Ω have the vertices

$$(0,0)\ ,\ (1,0)\ ,\ (1,1)\ ,\ (0,1)$$

f be defined by

$$f(x,y) = e^{5(x+y)}$$

and

$$\varepsilon = 10^{-2}\ .$$

Additional problems will be given out on Monday, 14 April, at 1:00pm.

If you have questions, contact:

Jim Boyce	Jacks 341	7-1658	home 858-1293
Jay Gischer	Jacks 450	7-3088	home 321-8643
Joe Oliger	Jacks 308	7-3134	home 321-6784

Spring 1980 - Comprehensive Programming Project

Final Set of Problems

Problem 3.

Let Ω have the vertices

$$(0,0)\ ,\ (1,0)\ ,\ (1,1)\ ,\ (0,1)\ ;$$

$f(x,y)$ be defined by

$$f(x,y) = |x-y|^{1/2}\ ;$$

and use $\varepsilon = 10^{-2}$.

Problem 4.

Let Ω have the vertices

$$(0,0)\ ,\ (2,0)\ ,\ (4,2)\ ,\ (4,3)\ ,\ (3,4)\ ,\ (2,4)\ ,\ (0,2)\ ;$$

define $\rho(x,y)$ by

$$\rho(x,y) = \begin{cases} 0 & \text{if } x^2+y^2 > 1 \\ \exp([x^2+y^2-1]^{-1}) & \text{if } x^2+y^2 < 1 \end{cases}$$

and then define

$$f(x,y) = \rho(2x-4\ ,\ 2y-4)\ ;$$

use $\varepsilon = 10^{-2}$.

Problem 5.

Let Ω have the vertices

$$(0,0)\ ,\ (1,0)\ ,\ (1,1)\ ,\ (0,1)\ ;$$

let ρ be as above in Problem 4, and define f by

$$f(x,y) = \rho(10x,10y) + \rho(10x-5\ ,\ 10y-15/2)\ ;$$

and use $\varepsilon = 10^{-2}$.

WINTER 1980/81 COMPREHENSIVE EXAMINATION

Numerical Analysis

(Subproblems have equal weight.)

Problem 1. [16 points]. Nonlinear Equations.
The equation $\sin x = x^2$ has two solutions, x = 0 and $x \doteq .87672$. The following fixed-point iteration schemes are proposed for finding the nonzero root.

$$\text{(a)} \quad x_{n+1} = (\sin x_n)^{\frac{1}{2}} \qquad x_0 = .5$$

$$\text{(b)} \quad x_{n+1} = x_n - \frac{(\sin x_n - x_n^2)}{(\cos x_n - 2x_n)} \qquad x_0 = 1.0$$

In each case, predict the limit (if any) of the sequence x_n, the order of convergence, and , if the convergence is linear, the asymptotic convergence factor. You may need to know that

$$\cos(.87672) \doteq .63967$$
$$\sin(.87672) = (.87672)^2 \doteq .76864$$

Problem 2. [13 points]. Interpolation.
The function $f(x) = \ln x$ is tabulated at x = 1, 2, 3, . . ., 100. The table shows four correctly rounded decimal digits after the decimal point. For what part of the table is linear interpolation sufficiently accurate to preserve the accuracy of the table?

Problem 3. [21 points]. Linear Systems.
Consider the system of n linear quations in n unknowns

$$(A + iB)(\vec{x} + i\vec{y}) = (\vec{c} + i\vec{d})$$

where A and B are real n × n matrices, $\vec{x}, \vec{y}, \vec{c}$, and $\vec{d}$ are real n-vectors, and $i = \sqrt{-1}$. We have two choices:

1. Solve the system by Gaussian elimination with partial pivoting, using complex arithmetic throughout.
2. Solve the real linear system

$$A\vec{x} - B\vec{y} = \vec{c}$$
$$B\vec{x} + A\vec{y} = \vec{d}$$

(a) Which is more efficient in terms of arithmetic operations performed? In terms of space used? Justify your answers.

(b) Assume that $A + iB$ is Hermitian (i.e. that $A = A^T$ and $B = -B^T$) and positive definite (i.e. that $(\vec{x} - i\vec{y})^T(A + iB)(\vec{x} + i\vec{y})$ is real and nonnegative for all real n-vectors $\vec{x}$ and $\vec{y}$, and vanishes only for $\vec{x} = \vec{y} = 0$). Reassess your answers to (a) in light of these assumptions.

(c) Solve, by any method you choose,

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} (\vec{x} + i\vec{y}) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

Problem 4. [10 points]. Numerical Stability.

(a) Show that if A is a nonsingular matrix, $\vec{y} = A\vec{x}$ and $\vec{y}' = A\vec{x}'$, then

$$\frac{\|\vec{y}' - \vec{y}\|}{\|\vec{y}\|} \leq \kappa(A)\frac{\|\vec{x}' - \vec{x}\|}{\|\vec{x}\|}$$

where $n(A) = \|A\|\|A^{-1}\|$, $\|\vec{x}\|$ is a vector norm, and $\|A\|$ is the subordinate matrix norm.

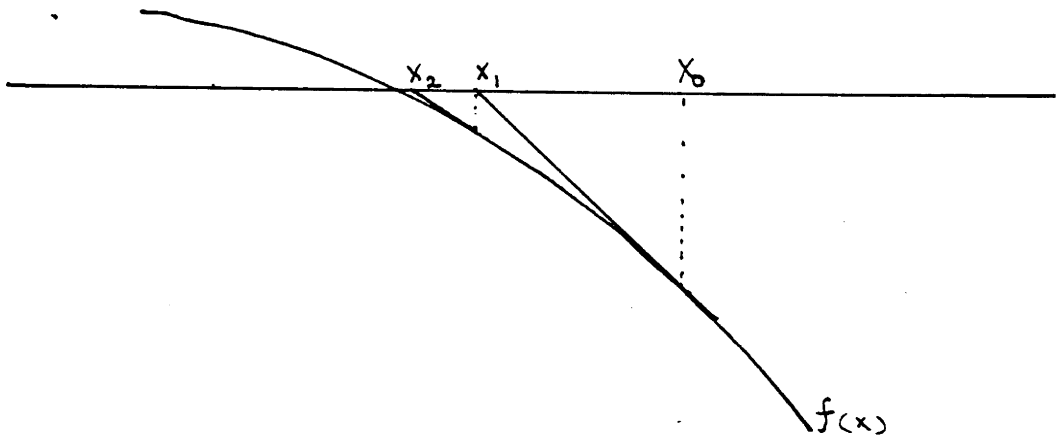
(b) It is claimed that if $\vec{x}$ is such that $\|A\vec{x}\| = C\|A\|\|\vec{x}\|$ where $C > \frac{1}{2}$, then relatively small changes in $\vec{x}$ produce relatively small changes in $\vec{y}$ no matter how large $\kappa(A)$ is. Give a proof or a counterexample.

NA Solutions.

1. (a) For $0 < x < x^* = .87672$, $\sin x > x^2$; whence $(\sin x)^{1/2} > x$; thus $x_n \to x^*$. Convergence is linear with asymptotic convergence factor

$$g'(x^*) = \frac{1}{2}\frac{\cos x^*}{(\sin x^*)^{1/2}} = \frac{1}{2}\frac{.63967}{.87672} = .365$$

(b) This is Newton's method. To see that $x_n \to x^*$ quadratically note that, if $f(x) = \sin x - x^2$, $f'(x^*) = \cos(x^*) - 2x^* \neq 0$ and $f'' < 0$ everywhere, so f is convex and Newton's method converges:



2. Interpolation error $\leq \frac{f''(\xi)}{8} = \frac{1}{8\xi^2} \leq \frac{1}{8i^2}$

for $i \leq x \leq i + 1$.

Roundoff error in table entries $\leq 5 \times 10^{-5}$.

So we want

$$\frac{1}{8i^2} \leq 5 \times 10^{-5}$$

or $i \geq 50$.

3. (a) I. Complex Gaussian elimination:

Storage: $2n^2$

Work (multiplies): $4 \cdot \frac{1}{3}n^3 = \frac{4}{3}n^3$

(adds): $2\left(\frac{1}{3}n^3\right) + 2\left(\frac{1}{3}n^3\right) = \frac{4}{3}n^3$

since a complex multiply can be done using 4 real multiplies and 2 real adds, and a complex add uses 2 real adds.

II. Real 2n x 2n.

Storage: $(2n)^2 = 4n^2$

Work (Mults.) $\frac{1}{3}(2n)^3 = \frac{8}{3}n^3$

(adds) $\frac{1}{3}(2n)^3 = \frac{8}{3}n^3$

Method I is twice as fast and uses half the space.

(b) Cholesky factorization can be used in either case, This doesn't change the comparison.

(c) $$\frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ -1 & -i & -1 & i \end{bmatrix} = I,$$

Therefore the solution is

$$\frac{1}{4}\begin{bmatrix} 7 \\ -2-i \\ 1 \\ -2+i \end{bmatrix}$$

4. (a) Since $y' - y = A(x' - x)$, $||y' - y|| \leq ||A|| \, ||x' - x||$.

Also $x = A^{-1}y$, so $||x|| \leq ||A^{-1}|| \, ||y||$; therefore

$$\frac{||y' - y||}{||y||} \leq ||A|| \, ||A^{-1}|| \, \frac{||x' - x||}{||x||}.$$

(b) As before, $||y' - y|| \leq ||A|| \, ||x' - x||$.

Divide by $||y|| = C \, ||A|| \, ||x||$ to get

$$\frac{||y' - y||}{||x||} \cdot \frac{1}{C} \cdot \frac{||x||}{||x||} \leq 2 \, \frac{||x' - x||}{||x||}.$$

Software Systems

1. Synchronization and communication (10 points)

A typical message-passing system might be based on two primitives:

- Send (process, message)
- **Receive** (process, ○ℳ••ⓒ℁ℳ①

where **Send** blocks until the message is queued for the receiver, but does not wait for a reply. In such a system, a remote procedure call might be implemented using those two primitives back-to-back -- i.e.:

```
proc RPC (process, message);
  begin
     Send (process, message):
     Receive (process, message):
  end;
```

What synchronization problems arise with this approach? How might you solve them?

Solution: Assume process PA executes RPC(PB,M1). Assume also that PA has no messages queued for it when process PB performs a Send(PA, M2). PA will then accept M2 as if it were a response to its request, M1. In general, this need not be the case. In particular, PB's Send may be part of an attempt by PB to execute RPC!

This problem can only be eliminated by re-defining the semantics of remote-procedure-call: For example, a unique transaction id can be generated for each outgoing call, such that only a reply containing that transaction id will be accepted as completing the call.

2. Paging (10 points)

Assume that we have a main memory that can hold 3 pages of size 1000 (decimal) words. The pager can take advantage of the fact that a page has not been modified since placed in main memory and will not cause a copy of that "clean" page to be sent to disk when that page is reclaimed. We are given the following reference string:

```
1000(r), 234(r), 3345(r), 805(w),
2998(r), 3768(r), 1002(w), 5806(w)
```

The numbers are word addresses. The (r) means read access, and (w) means write access. Assume that the main memory is originally empty. Give the sequence of paging operations that would be performed assuming an LRU page replacement algorithm. Give your answer in terms of SWAPIN(i) or SWAPOUT(j) where *i* and *j* are page numbers.

Solution: To simplify things, change everything to page references:

```
1( r) SWAPIN(1)
0(r)  SWAPIN(0)
3(r)  SWAPIN(3)
0(w)  nothing
```

```
2(r)    SWAPIN(2)    (no need to swapout 1)
3(r)    nothing
1(w)    SWAPOUT(0)
        SWAPIN(1)
5(w)    SWAPIN(5)    (no need to swapout 2)
```

3. Multiprocessing (9 points)

Assume that Progressive Computers Inc. has decided to go from running its programs on a single machine to a multiprocessor configuration with shared memory. Since their programs always ran in a multiprogramming environment they expect very few problems in converting to multiprocessors.

a. (3) Give a short list of feasible benefits they can expect to reap from this change. Include a brief explanation of **each** benefit.

Solution: Increased reliability due to redundancy of processors. Increased performance through parallelism and load sharing. The ability to handle increased complexity due to modular decomposition of tasks into subtasks capable of being executed on multiple processors simultaneously.

b. (3) Given a configuration of exactly two processors, why will it be in general impossible to expect twice the processing power?

Solution: All the synchronization necessary in memory and data base access.

c. (3) Assume they use a simple primitive such as a test-and-set operation to synchronize processes in a multiprogramming environment. Give the one (possibly fatal) flaw in the architecture of their synchronizing primitive that would cause it to work in a multiprogramming environment, but not in a multiprocessing environment.

Solution: Not locking out the memory bus access to the other processor when altering a lock.

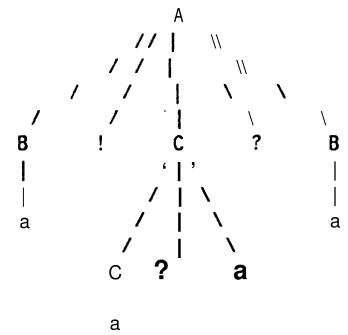
4. Parsing (10 points)

Consider the following BNF grammar

```
<A> ::= <B> ! <C> ? <B>
<B> ::= a | <B>! a
<c> :: = a | <C> ? a
```

a. (5) Show the parse tree for a!a?a?a.

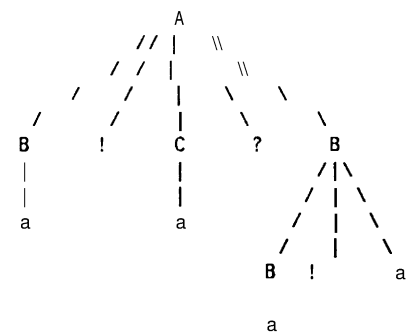
Solution:



b. (5) This grammar could not be used for operator precedence parsing because in some cases ! has greater precedence than ?, and in some cases the reverse is true. Which case occurs in the sentence of part a? Give a sentence and parse tree which illustrate the other case.

Solution: In the sentence in part a, ? has higher precedence than !, because a?a must be reduced to C before ! can be used in a reduction.

In the string a!a?a!a, ! has higher precedence than ?.



5. Binding time (12 points)

Binding is the association of some attribute with a name. For each point below, give an example of a programming language that involves binding at that time. Be specific about the language and what is being bound.

a. (2) compile time

b. (2) link time

c. (2) load time

d. (2) block entry

e. (2) procedure call

f. (2) assignment (give an example where some attribute besides value is bound at assignment)

Solution: (There are many possible answers - these serve as examples.)

a. Compile time -- types in Algol, Pascal, Fortran; array size in Fortran, Pascal.

b. Link time -- procedure, function, or subroutine correspondences in any language with separate compilation (e.g. Fortran, PL/1); external names in PL/1; COMMON blocks in Fortran.

c. Load time -- absolute addresses of code for most languages; of variables in Fortran and other languages with static memory allocation.

d. Block entry -- size of variably-dimensioned arrays in Algol, PL/1.

e. Procedure call -- correspondence between actual and formal parameters (any language that allows parameters for procedures).

f. Assignment -- type in LISP, SNOBOL, APL.

6. Interpreters (9 points)

LISP is usually implemented by an interpreter rather than a compiler. Give three characteristics of LISP that are related to this fact (for example, features that would be harder to implement with a compiler).

Solution:

a. Types of variables are determined (and can be modified) at run time.

b. Data computed by the program can be executed as code.

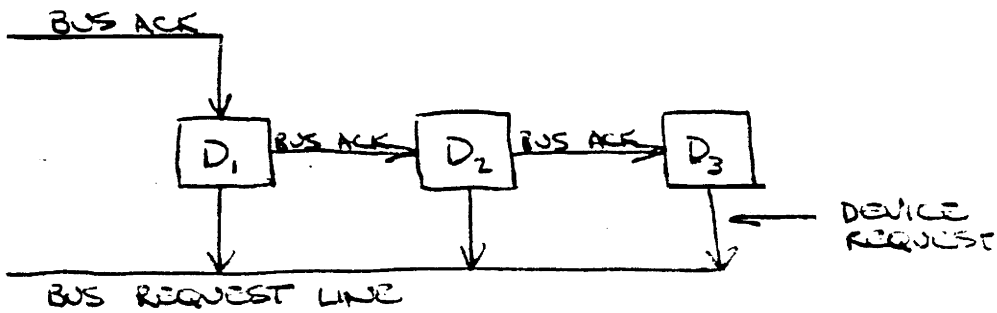
c. Variables are bound dynamically at procedure-call time.

Hardware Systems

1. (12) Bus **communication**

   a. (8) Describe how bus arbitration may be accomplished via a centralized daisy-chain technique, indicating clearly all the essential control signals required. Illustrate your answer with a block diagram of a single bus system with 3 devices on the bus.

Solution: Signals in a daisy chain are *bus request* and *bus acknowledge*. Whenever a request occurs the bus devices are given the opportunity to use the bus in chain-order.

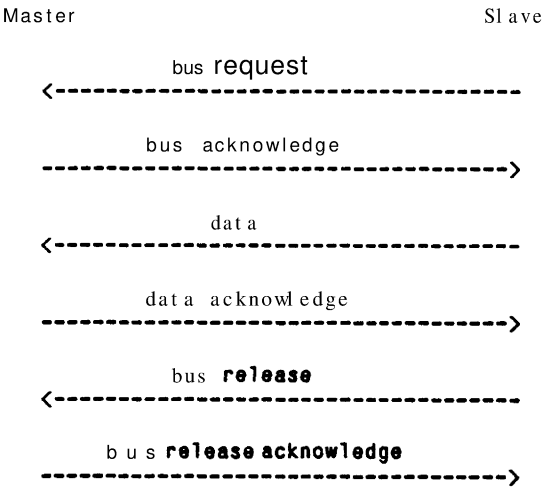


Device D_i does the following: if *bus acknowledge* and D_i has an outstanding request then use bus, else send the acknowledge to D_{i+1}. A request is held high by D_i until it receives an acknowledge.

   b. (4) What is meant by *fully-interlocked handshaking* in bus communication? Illustrate your answer with a simple timing diagram.

Solution: Fully interlocked handshaking means that both commuicating parties send acknowledgments.

A typical situation might be:



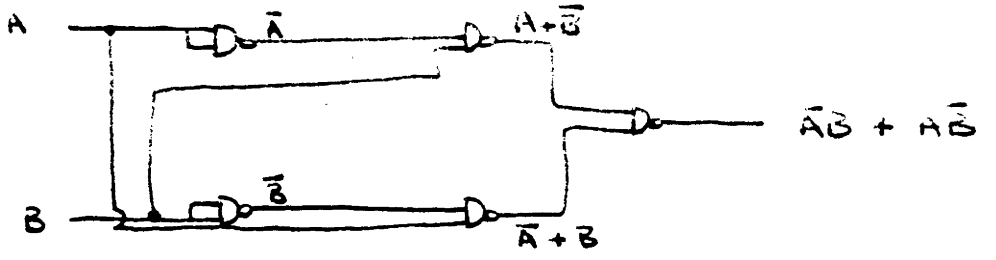
2. (10) Logic diagrams

Draw the logic diagram of an exclusive-OR function of two inputs using NAND-gates.

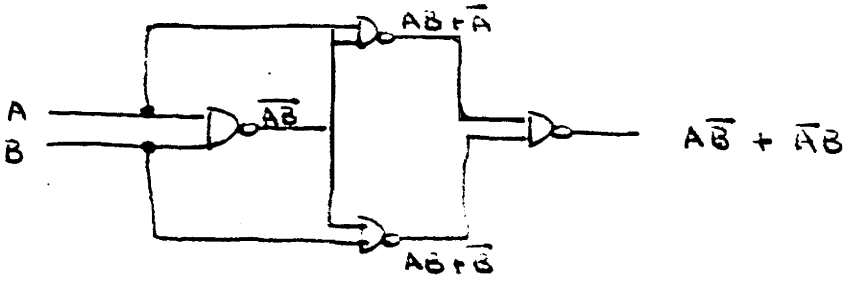
Solution: Exclusive-OR is:

$$\bar{A}B + A\bar{B} = \overline{\overline{\bar{A}B} \cdot \overline{A\bar{B}}} = \overline{(A+\bar{B})\cdot(\bar{A}+B)}$$

NAND is: $\overline{A\bar{B}} = \bar{A} + \bar{B}$



BETTER YET:

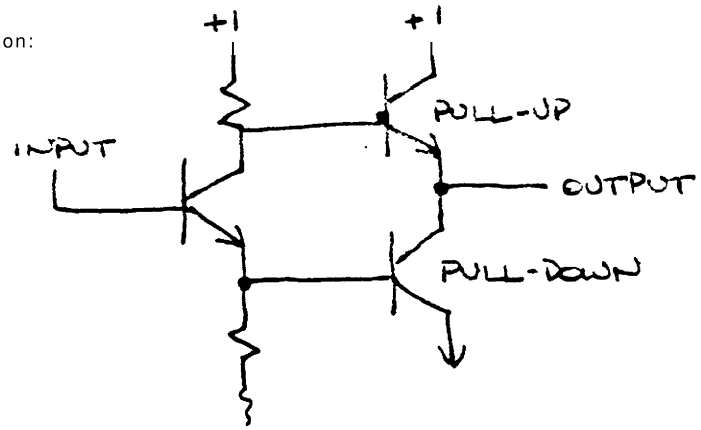


3. (8) TTL logic

Describe the following terms with respect to TTL gates:

a. totem-pole output

Solution:



Yields a faster switching time since pull up/down is through a transistor

b. tri-state output

Solution: Output can be in three states: low, high, or off. In the off state the output is free to drift if another device on the same line sources or sinks it. Especially useful for busses.

c. fan-out

Solution: Number of loads an output can drive. Each device input may use one or more loads.

d. noise margin

Solution: Difference between the highest low output and the lowest high output (o r switching threshold). It determines the susceptibility to noise.

4. (10) Cache memory

In a cache-memory system, let:

cache access time, t = 100 nsec
main memory access time, T = 1 microsec
block-size, B = 8 words
main-memory-to-cache connection size, C = 2 words

hit-ratio for memory access, H = .9

a. (3) What is the effective memory access-time, if a read-through policy is used?

Solution:

EAT = <cache access time>*<hit rate> + (1-<hit rate>)*<memory access time>
= t*H + (1-H)*T
= 100 • .9 + .1 • 1000
= 190 nsec

b. (5) What is the effective memory access-time, if no read-through policy is assumed, so that the words in a block are fetched strictly sequentially on a miss and then access from the cache?

Solution: Without read-through the delay is:

EAT = <cache access time>*<hit rate> +
(1-<hit rate>)*(<cache fill time> + <cache access time>)

cache-fill-time, F = (<block size>/<cache connection size>)*
<memory access time>
= (8/2)*(1000 ns) = 4000 ns

EAT = t*H + (1-H)*(F + t)
= 90ns + .1 • (4000ns + 100 ns) = 500 ns

c. (2) How many comparators are needed if the cache size is 16K words?

Solution: The number of comparators is <cache size>/<block size> since only one comparator is need per block:

$2^{14}/2^3 = 2^{11}$ = 2K comparators

5. (20) Computer organization

You are given a machine architecture with the following hardware:

- 16-bit words (all instructions operate on words)
- a hardware stack
- an ALU
- 2^8 bytes of memory (byte addressable)

- a single fixed size instruction format

There are two memory access instructions:

- push <addr>
- pop <addr>

which cause data to be moved from memory to/from the stack.

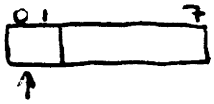
There are several address modes:

- absolute address: <addr> is a memory location
- direct: the top of stack contains the location (it is □□□□)
- indexed: top of stack (it is popped) + <addr>

There are 60 oiher O-address instructions which perform operations on the stack.

a. **(4)** Give an instruction encoding which minimizes the sizes of instructions in byte increments.

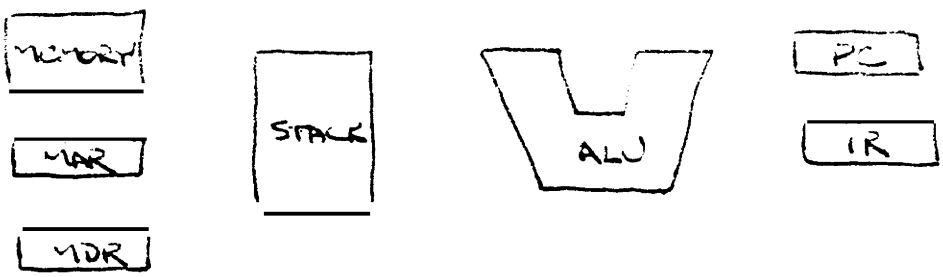
Solution: Something like the following will work:



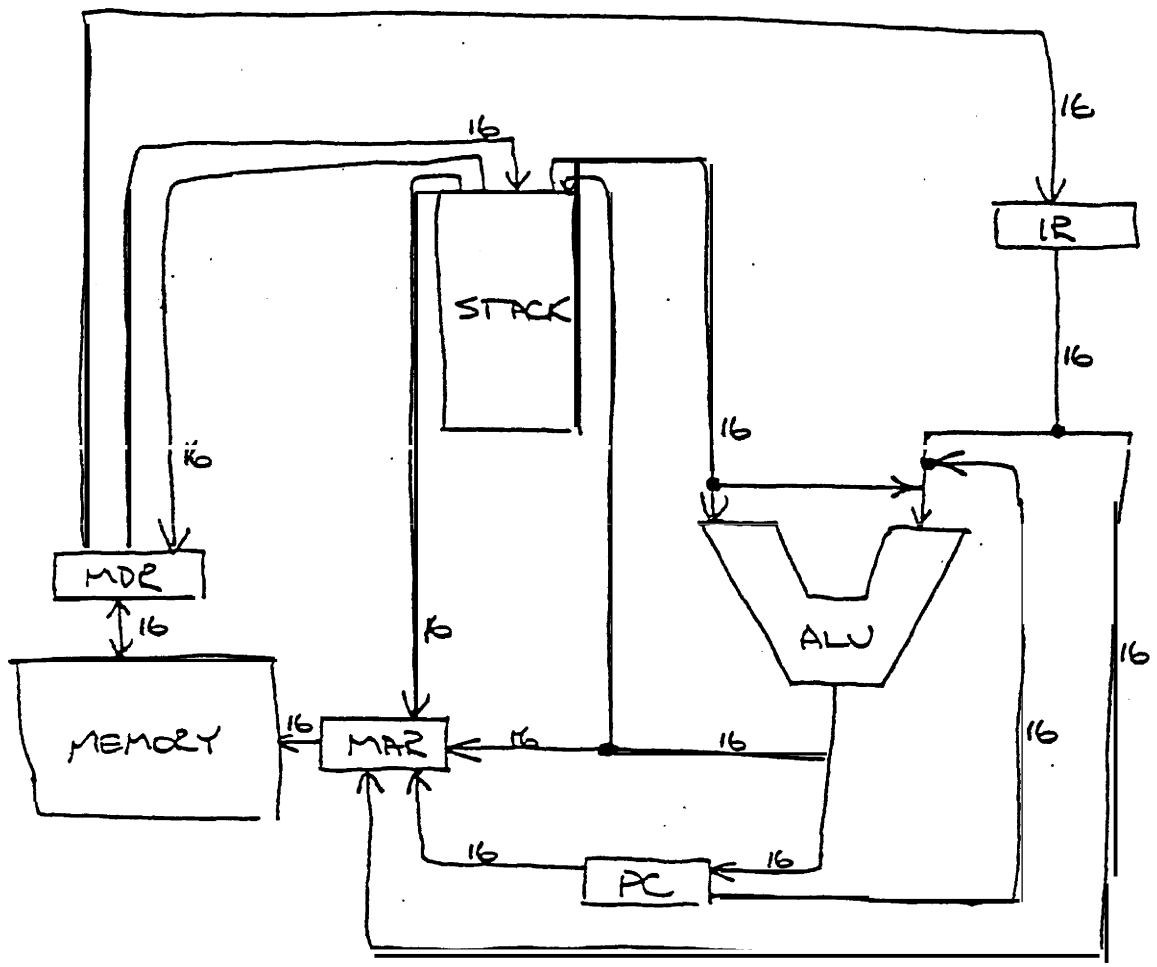
00 - not push/pop
01 - push
10 - pop

If not push/pop, the remaining bits encodes the other instructions. For push/pop there are 2 bits to encode the three addressing modes. If the address mode is 1 or 2 then a byte follows, otherwise it is unused.

b. (8) Using the following blocks draw a block diagram of the organization showing all data paths and indicating their sizes and direction of data flow.



Solution:



c. (8) Using the notation:

A --> B : description

to describe data flow from A to B, show the fetch, decode, and execution cycle for the instruction:

```
push indexed y
```

where y is the offset. The sequence starts with:

```
PC --> MAR : instruction address to memory
```

There is no need to show control, but the descriptions can indicate operations that occur.

Solution:

PC --> MAR:	instruction address to memory
MAR --> memory:	address to memory
memory --> MDR:	instruction to MDR
MDR --> IR:	Instruction to IR
stack --> ALU:	pop stack into ALU
IR[8:15] --> ALU:	offset part (y) into ALU
ALU --> MAR:	after add send new object address
MAR --> memory:	operand address
memory --> MDR:	operand value
MDR --> stack:	push the value
PC --> ALU:	send PC to ALU
ALU --> PC:	after incrementing by 2

ARTIFICIAL INTELLIGENCE

1. *Searching with Lisp (24* points)

Consider the following LISP program given, for your convenience, in both LISP external notation and in MACLISP.

findpath[*x, y*] ← *fp* 7 [⟨*x*⟩, *y,* NIL]

fp 1[*u, y, path*] ←
if n *u* **then** LOSE
else if a *u* ε *path* **then** *fp1*[**d** *u, y, path*]
else if a *u* = *y* **then** *reverse*[*y* . *path*]
else [λ*w:* if *w* = LOSE **then** *fp1*[**d** *u, y, path*] **else** *w*]
[*fp1*[*successors* [**a** *u*], *y,* **a** *u* . *path*]]

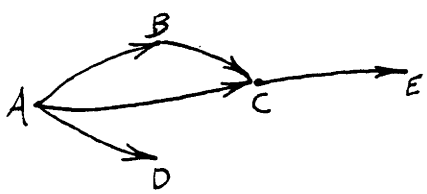
```
(defun findpath (x y) (fp1 (list x) y nil))

(defun fp1 (u y path) (cond
        ((null u) 'lose)
        ((member (car u) path) (fp1 (cdr u) y path))
        ((equal (car u) y) (reverse (cons y path)))
        (t ((lambda (w) (cond
                ((eq w 'lose) (fp1 (cdr u) y path))
                (t w)))
            (fp1 (successors (car u)) y (cons (car u) path))))))
```

The program searches (depth-first) for a path from x to y in a finite directed graph in which the successors of the node x are given by the function *successors.* When the search is successful the value of *findpath*[*x* , y] is a list of nodes starting with x and ending with y such that each node except x is a successor of the preceding node. When the search is unsuccessful *findpath*[*x , y*] = LOSE.

Assume that the cost of finding a path is dominated by the cost of computing successors of x.

a. (12 points) How is the above algorithm inefficient? Give a simple example of its inefficiency.

Solution: Because the program remembers only nodes on the path it is presently searching it can recompute the successors of a node that can be reached on different paths. An example is the graph



Given that *successors*[A] = (B C D), *successors*[C] will be computed twice.

b. (12 points) Write a more efficient LISP program to perform depth-first search. Remember the assumption about costs. You may use external notation or MACLISP or INTERLISP notation.

Solution:

findpath[x,y] ← [Xw. if a w = LOSE **then** LOSE **else** *reverse w*]
[*fp1*[⟨x⟩, y, NIL, NIL]]

fp1[*u, y, path, seen*] ←
if n *u* then LOSE . seen
else if a *u* ε seen then *fp1*[**d** *u, y,* path, seen]
else if a *u* = y then *path*
else [Xw. if a w = LOSE then *fp1*[**d** *u, y, path,* **d** *w*]]
[fp 1 [successors a *u, y,* a *u* . *path,* **a** *u* **. seen]]**

2. *Quickies* (12 points)

For these questions, a few phrases to indicate your understanding will suffice. (3 points each)

a. Why does the speech understanding problem require techniques from both AI and pattern recognition?

Solution:

The input to speech understanding systems is noisy and incomplete. This makes statistical methods from pattern recognition desirable.

Knowledge sources for speech understanding include models of the semantics and pragmatics of the utterance, making "knowledge representation" techniques from AI desirable.

b. What are the basic ways in which RSTRIPS & ABSTRIPS are improvements on STRIPS?

Solution:

RSTRIPS uses a goal protection system to handle the problem of sub-goal interaction.

ABSTRIPS plans more efficiently than STRIPS by planning hierarchically, putting operator pre-conditions in order by importance and difficulty.

c. Relate the problems of unification and of simple pattern matching.

Solution:

Pattern matching is a kind of unification in which one of the formulas has only constants.

d. What are a few of the things which make "story understanding" hard for computers?

Solution:

Story understanding requires real world knowledge, such as physical world relationships and human goals, which are hard to give to a computer. Related to this is the need for natural language parsing abilities, including the ability to resolve pronoun references.

3. Constraint Application (9 points)

Indicate the kinds of prior constraints applied in

- case analysis of sentences;
- blocks-world vision;
- speech understanding;

(three constraints in each area are sufficient for full credit)

Solution:

- case analysis:
  - The possible meanings of sentence verbs constrain the cases of noun groups in the sentence;
  - The case of a noun group constrains the main noun of the group;
  - The preposition of a prepositional phrase constrains the case of the noun group in the phrase;
  - Sentence position constrains the case of noun groups;
  - The case of a noun group constrains the cases of the other groups in the sentence;
  - Sentence context constrains noun & verb group meanings;

- blocks-world vision:
    - Line labellings constrain trihedral vertex labellings;
    - Illumination and shadows constrain line labellings;
    - Knowledge of boundaries constrains line labellings;
- speech understanding:
    - characteristics of speech sounds;
    - consistency in pronunciation;
    - stress and intonation patterns in speech;
    - grammatical structure of language;
    - meanings of words and sentences;
    - the context of conversation;

4. *Representation* (15 points)

Consider the following sentences:

Volcanos in the US. are generally dormant.
Mount Saint Helens is the only Volcano in Washington.
Volcanos are mountains.
Mountains are geological features.
Washington is in the US.
A volcano in Washington erupted recently.

a. (5 points) Express these sentences in a frame-like notation such as the "delineation units" described by **Nilsson.**

Solution:

```
x | US-volcano
        self : (subset-of volcanos)
        location : US
        condition : DORMANT

Mount-Saint-Helens
        self : (only-element-of VinW)
        location : (is-in Washington)

x | VinW
        self : (element-of volcanos)
        location : (is-in Washington)

x | volcano
```

```
        self : (element-of mountain)

x | mountain
        self : (element-of geological-features)

Washington
        location : (is-in US)

Volcano-A
        self : (element-of volcanos)
        location : (is-in Washington)
        condition : ACTIVE
```

b. (5 points) Why might units notation be used instead of First Order Logic in some situations? Why in general might one representation be used instead of another with equal or greater expressive power?

Solution: Units notation is somewhat more modular than First Order Logic, has a more uniform structure, and is better suited to default reasoning. In general, different representations are used when their expressive power is best suited to the application, and because they may encode more heuristics for deductive operations.

c. (5 points) Consider the questions

Is Mount Saint-Helens a geological feature in the U.S.?
Is Mount Saint-Helens dormant?

What kinds of rules are necessary in order to deduce heuristically reasonable answers to these questions from the units you indicated above?

Solution: Rules that encode the property inheritance characteristics of "element-of", "only-element-of" "subset-of", and the transitivity of "is-in" are necessary.

Rules which handle default reasoning on a hierarchy are necessary, e.g., that override inheritance of the "DORMANT" property by Mount-Saint-Helens with the particular knowledge of its activities in Washington. In this case the "condition" slot is implicitly default, as might be all slots that are not "is-a" links.

Problem 1. [20 points]. A mediocrity queue is a data structure that dynamically maintains a set S of numbers and executes a sequence of instructions $I_1, I_2, I_3 \ldots, I_{,,} \ldots$. Each I_i is either one of the following forms:

insert$[x]$ (meaning S $\leftarrow$ S $\cup \{x\}$)

delete$[x]$ (meaning S $\leftarrow$ S $- \{x\}$)

getmedia- (return the value of the median of S).

The set S is initially the empty set, and only distinct elements will be kept at any time. The median of S is the $\lceil |S|/2 \rceil$-th smallest number in S.

(A) [3 points]. Give an implementation of a mediocrity queue such that i_n = O(n), d_n = O(n) and g_n = O(1); i_n, $d_{,,}$ g_n are the respective worst-case costs of executing an insertion, a deletion, and a getmedian when $|S| = n$.

(B) [12 points]. Repeat (A) with $i_n = O(\log n)$, $d_n = O(\log n)$, and $g_n = O(\log n)$.

(C) [5 points]. suppose a mediocrity queue is available, such that the total cost of executing any sequence of n instructions is f(n). Give an algorithm that sorts n distinct numbers in time $f(3n - 1)$ + O(n), by making use of the mediocrity queue.

Remarks. In the solutions to parts (A) and (B), give only a high-level description for standard data structures, but it should contain enough information to justify the asserted performance. For example, you can "maintain a 2-3 tree under insertions with O(log n) cost per insertion", but you cannot "maintain some kind of hash table that has a cost $0(\sqrt{n})$ per insertion".

Problem 2. [30 points]. Let $x_1, x_2, \ldots, x_n$, m be $n + 1$ input real numbers that are all positive and distinct.

(A) [5 points]. Give an O(n log n)-time algorithm for deciding if there exist distinct i, j such that $x_i + x_j =$ m.

(B) [lo points]. Give an $O(n^2)$-time algorithm for deciding if there exist distinct i, j, k such that $x_i + x_j \cdot x_k = m$.

(C) [15 points]. Give an $O(n^2$ log n)-time algorithm for deciding if there exist distinct i, j, k, ℓ such that $x_i +_{,} x_j \cdot x_k \cdot x_\ell = m$.

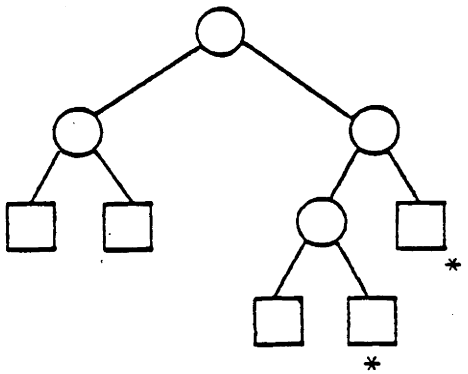
Remark. We are using a random-access computer that can perform infinite-precision real arithmetic. (You may ignore overflow problems on such machines.)

Problem 3. [10 points]. A *k-right-biased binary* tree is a rooted binary tree such that any path from the root to a leaf takes right branches at most k times.

(A) [2 points]. What is the maximum number of leaves that any 2-right-biased binary tree with height 4 can have?

(B) [8 points]. What is the maximum number of leaves that any 2-right-biased binary tree with height h can have ($h > 0$ an integer)?

Remark. We show below a 2-right-biased binary tree with height 3 and 5 leaves. This is not a l-right-biased binary tree as the * leaves show.



Algorithms and Data Structures

Problem 1.

(A) Perhaps the simplest idea is to store the elements of the queue in a sorted (increasing) array. Insertion requires time linear in the the size of the queue to move the elements of the array around to make room for the new element. Deletion requires linear time to move the the elements of the array to fill in the gap created by the deletion. The site of the insertion or deletion can be found either using a linear scan of the array, or by doing a binary search. To find the median in constant time, it is necessary to keep another variable holding the current size of the queue. It can be updated in constant time during an insertion or deletion. The median is found by looking in the $\lceil |S|/2 \rceil$-th element of the array.

(B) Use your favorite flavor of a balanced search tree, e.g. AVL tree, 2-3 tree, or R-B tree, to maintain a sorted list with logarithmic insert and delete times. In addition, keep in each node the weight of the subtree hanging from that node. This will permit you to find the median in logarithmic time by looking at the weights (and weights of siblings) along a single path from the root.

(B′) This can be improved to a method that still does insertion and deletion in logarithmic time, and finds the median in constant time. The idea is to keep the median in a specific location, so that it is easy to End. The elements greater than the median are kept in a balanced search tree and the elements less than the median are kept in another one. Insertion and deletion require a comparison to see which tree is effected, and the logarithmic time to perform the operation. If the operation changes the median, the old median is inserted into the appropriate tree as an extreme value, and the new one is deleted from the other tree. Both of these can be done in logarithmic time.

(C) Here is a way to sort n distinct numbers using just $3n - 1$ mediocrity queue operations. First, insert all n numbers into the queue (n insertions). Then, alternately find the median and delete it until you have found each element as the median once (n getmedians and $n - 1$ deletions). After an element is found as the median, it is inserted into the right place in an array in constant time. (The first one goes into the $\lceil |S|/2 \rceil$-th location. Thereafter, alternately the medians of the queue will be the first element larger or smaller than the part of the array that has been sorted.)

Problem 2.

(A) The first step is to sort the set $\{x_i\}$. This takes time $O(n \log n)$. Then, for each of the x_i, search the sorted table for the value $m - x_i$. Using binary search does each of the searches in time $O(\log n)$ and the entire step in time $O(n \log n)$. After finding the right value in the table, it is necessary to check that the proposed values of x_i and x_j are different. This is done (at most) once for each search.

(A′) Once you have the sorted table, it is possible to search it for a pair whose sum is m in time $O(n)$.

```
i := 1;
j := n;
while i < j do
    begin
    if x_i + x_j = m then exit loop with success;
    if x_i + x_j < m then i := i + 1;
    if x_i + x_j > m then j := j - 1;
    end;
```

In effect, this program takes the two sets $\{x_i\}$ and $\{m - x_j\}$ both of which are sorted, and merges them to see if they have an element in common.

Note: Several people said to sort the inputs with an $O(n \log n)$ sorting algorithm, e.g. quicksort. While quicksort does have an average running time of $O(n \log n)$, its worst-case running time is $O(n^2)$.

(B) First, sort the inputs. Then, for each of the x_i use a simple modification of (A') to see if $m - x_i$ occurs as the product $x_j x_k$. The sorting is done once in time $O(n \log n)$ (Actually, an $O(n^2)$ sort is sufficient.) Then a linear scan of the table is performed n times for a runtime of $O(n^2)$. The little care needed to make sure the solution uses distinct values at most multiplies the running time by a constant.

Note: Many people felt that the products could be generated in order in time $O(n^2)$. Unfortunately, this takes time $0(n^2 \log n)$.

(C) The important observation in that it takes time $O(n^2 \log n)$ to sort n^2 numbers. One solution is first to sort the set of products $\{x_k x_l : k < l\}$. Then, perform n^2 binary searches of that table looking for the values $\frac{m-x_i}{x_j}$. Each search takes $O(\log n)$ time. At most two of the products can have x_i or x_j as factors. This means that there is O(1) work to see if a there is actually a solution after each successful search.

(C') Another solution is to sort the sets $\{\frac{m-x_i}{x_j} : i \neq j\}$ and $\{x_k x_l : k < l\}$. Check these two sets for a common element by merging them. A little care is necessary to make sure the values in the solution are all distinct. When a common value is found. an element of the first set can "collide" with at most two elements in the set. This means that only O(1) work is needed to for a solution if when a common value is found.

Problem 3.

(A) 11. There are two definitions height that give values that differ by one. Some people count the number of nodes (including the root) on the longest path in the tree. Others count the number of edges. The *diagram* in the question showed which definition to use.

(B) A maximal k-right-biased binary tree d of height h consists of a maximal k-right-biased binary tree of height $h - 1$ hanging to the left of the root and a maximal $k - 1$-right-biased binary tree of height $h - 1$ hanging off to the right. A O-right-biased binary tree consists of a single path with a single leaf. A maximal 1-. . . tree of height h consists of h of those 0-. . . trees and a 1–. . , tree of height 0. So a maximal 1–. . . tree has $h + 1$ leaves. A maximal 2-. . . tree consists of h of those 1–. . . trees and a tree of height 0. The total number of leaves is

$$1 + 2 + 3 + \cdots + (h - 1) + h + 1 = \frac{h(h+1)}{2} + 1.$$

Problem 1. [10 points]. A *k-wheel* is an undirected graph on $k+1$ vertices $v_0, v_1, \ldots, v_{k-1}, u$ with edges $\{v_i, v_{(i+1) \bmod k}\}$ and $\{u, v_i\}$ for $0 \leq i < k$; u is called the center. Prove that the following problem is NP-complete: Given a graph G and positive integer k, determine if G contains a k-wheel.

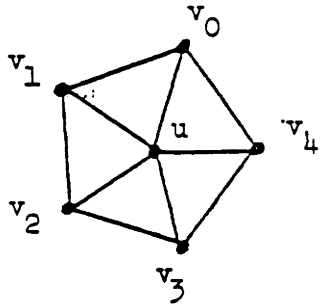


Figure. A S-wheel.

Problem 2. [25 points]. Let the array A be such that initially

$$\forall i.(0 \leq i \leq n \supset A[i] = 1). \tag{1}$$

(a) [only 5 points] Write a flow chart type program (i.e. with assignments and go tos), not using multiplication or any array other than A, that terminates with

$$\forall i.(0 \leq i \leq n \supset A[i] = \binom{n}{i}, \tag{2}$$

i.e. it computes a vector of binomial coefficients. Remember the recurrence relation of Pascal's triangle which may be written

$$\binom{n}{k} = \text{if } k < 0 \vee k > n \text{ then } 0 \text{ else if } k = 0 \vee k = n \text{ then } 1 \text{ else } \binom{n-1}{k-1} + \binom{n-1}{k}, \tag{3}$$

but it shouldn't be used directly as a recursive program, because it recomputes the $\binom{i}{k}$ so often that it takes exponential time.

(b) (20 whole points] Attach sentences of first order logic to each label of your program so partial correctness as expressed by attaching equation (2) to the exit label can be proved by the method of invariant assertions. We just want the assertions — not the proof.

Problem 3. [10 points]. M is a machine which takes its input from a papertape-like file (read-only, left-to-right) and prints an acceptance of certain input tapes. Apart from a finite-state control, its only memory is a pushdown stack, with the usual operations of push, pop, and test top symbol. An unpoppable internal state S of M is one in which the stack can never become shorter than it is in S, whatever the input. (In other words, the current stack symbols will never be popped.) Is there an algorithm to recognize such states? (Describe one or show undecidability.)

Problem 4. [15 points]. Let $L_{ij} (i \geq 0, 1 \leq j \leq 2)$ be the set of languages recognizable by machines with i counters as the only unbounded memory, and with left-to-right input if $j = 1$, two-way input if $j = 2$. (Counters can be incremented' dccremcnted, and tested for zero.) What inclusion relations hold among the L_{ij}'s? (State reasons briefly; detailed proofs are not required.)

Solutions - Mathematical Theory of Computation

1. Solution to the "k-wheel" problem.

We reduce the Hamiltonian circuit problem to the k-wheel problem. Given a graph $G = (V, E)$ on n vertices, one can clearly construct in polynomial time the graph $H = (V', E')$, where $V' = V \cup \{w\}$ and $E' = E \cup \{\{w, v\} \mid v \in V\}$. The following result then completes the reduction.

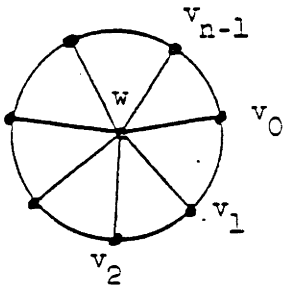
Theorem. G has *a Hamiltonian* circuit if and *only if* H contains an n-wheel.

Proof.

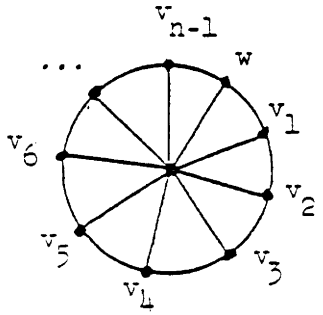
(A) If G has a Hamiltonian circuit $v_0, v_1, v_2, \ldots, v_{n-1}$ then H contains an n-wheel with the set of edges $\{\{v_i, v_{(i+1) \bmod n}\}, \{w, v_i\} \mid 0 \leq i < n\}$.

(B) If H contains an n-wheel:

Case 1. w is the center: Clearly, the rest of the wheel gives a Hamiltonian circuit for G.



Case 2. $V_0 \neq$ w is the center: Let us label the vertices as shown, then $v_0, v_1, v_2, \ldots, v_{n-1}, v_0$ is a Hamiltonian circuit for G.



This completes the proof of the theorem. □

Solution to Problem 2.

Verification question.

Program:

```
          i := 1;
oloop:    if i=n  then go to end;
          i  := i+1;
          j := i-1
iloop:    if j<1 then go to nexti;
          a(j) := a(j) + a(j-1);
          j  := j-1;
          go to iloop;
nexti:    go to oloop;
  end:    return
```

Assertions: The following sentences apply before execution of the statements to which they are attached.

oloop: $(\forall k)\left[(0 \leq k \leq i \supset a(k)=\binom{i}{k}) \wedge (i<k \leq n \supset a(k)=1)\right]$

iloop: $(\forall k)\left[(0 \leq k \leq j \supset a(k)=\binom{i-1}{k}) \wedge (j<k \leq i \supset a(k) = \binom{i}{k}) \wedge (i<k \leq n \supset a(k)=1)\right]$

end: $(\forall k)(0 \leq k \leq n \supset a(k) = \binom{n}{k})$ (output assertion)

input assertion: $(\forall k)(0 \leq k \leq n \supset a(k)=1)$

3. M is a machine which takes its input from a paper tape-like file (read only) and prints acceptance of certain data tapes. Apart from a finite-state control, its only memory is a pushdown stack, with the usual operations of push, pop, and test-top-symbol. An unpoppable internal state S of M is one in which the stack can never become shorter than it is in S, whatever the input, Is there an algorithm to test wuch states?

Answer: A given state S of M can be modified into the initial state of a machine M' that accepts those inputs which make M's stack shorter than it is in S. Standard methods construct a CF grammer for this language. Equally standard methods test the language for emptiness. (An alternate method of proof defines the set of unpoppable states recursively, by a monotone recurrence,)

4. Answer: (i=0) L-to-R and two-way finite state machines recognize finite-state languages. A one-counter machine recognizes only recursive sets, while a two-counter machine CL-to-R or not) is universal. An L-to-R one-counter machine can't recognize

$$\{a^i b^j c^i \} \cup \{a^i b^j d^j\} ,$$

while a two-way one-counter machine can. Therefore,

$$L_{01} = L_{02} \quad L_{11} \subset L_{12} \quad L_{21} = L_{22} = \text{everything else .}$$

Spring 1980/81 COMPREHENSIVE EXAMINATION

ALGORITHMS AND DATA STRUCTURES

Problem 1. [20 points]. We wish to design a data structure that deals with objects, each of which has a value. Many objects can have the same value. Specifically we wish to support the following operations:

(1) Creation. Given an array of objects and the size of the array, create a data structure containing exactly those objects, which supports the operations of deletion and query defined below.

(2) Deletion. Given an index to the array of objects, delete the corresponding object from the data structure.

(3) Query. Answer the question: "Do all (remaining) objects in the data structure have the same value?"

The operations of deletion and query are being done in real time. Therefore, the most important property of this data structure is that the slowest of the operations of deletion and query be as fast as possible in the worst case. Subject to this constraint, the expected time for the creation operation should be as fast as possible.

Describe a data structure and the algorithms for implementing the three operations. Estimate the time required for each operation. Justify any estimates that are not obvious.

For full credit, the time required for the deletion and query must be constant in the worst case, and the expected time for the creation operation must be 0(n), where n is the initial size of the input array. Partial credit will be awarded for slower solutions. Specifically, if the deletion and query operations require constant time, but creation requires $O(n \log n)$ expected time, three quarters credit (15 points) will be awarded.

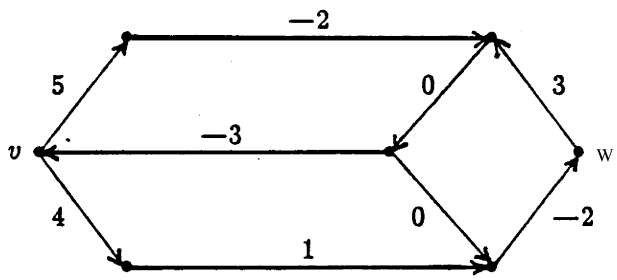
Problem 2. [20 points]. Someone wishes to generate bad binary search trees quickly, given a set of keys and a distribution of the expected frequency of search keys. There are limits to how bad the trees can be, however. Specifically, each key must appear in the tree exactly once, and the tree must have the required order property. That is, that if the tree is traversed in symmetric order (inorder) the keys are reached in alphabetical order. When searching for a key, the probability that it is actually in the tree is negligible, and the probability that it is between any adjacent pair of keys in the tree is known.

That is, the input consists of n, the number of keys in the tree, the n keys, and an array, $freq[0 \ldots n]$. If we let key_0 denote $-\infty$ and let key_{n+1} denote $+\infty$, then $freq[i]$ contains the frequency with which the sought key will be between key_i and key_{i+1}. The tree that is desired will have the longest external path length, weighted by the entries of $freq$.

(a) [15 points] Design a polynomial time algorithm to find the tree with the worst possible expected search under the assumptions above and show that your algorithm works. Make your algorithm asymptotically as fast as possible. (Hint: you may use without proof the fact that in this tree no key has two non-null sons.)

(b) [5 points] How fast does your algorithm run? Justify your estimate. Express your answer as $O(f(n))$, for some suitable $f(n)$.

Problem 3. [20 points]. Let G be an directed graph, with a weight, which may be any integer (positive, negative or zero), given for each edge. For a given vertex v, we define a zero-cycle to be a path starting and ending at v, passing through at least one other vertex, such that the sum of the edge-weights along the cycle is zero. No vertex may appear more than once along such a cycle (except for the initial vertex which appears only at the beginning and end). For example, in the graph shown below, v has a zero-cycle, but w does not. Show that the problem of determining whether G has any zero-cycles is NP-complete.



Problem 1. We will create a new array of size n, where the element corresponding to each position in the input array is a pointer to a counter containing the number of instances of that value. There is also a global counter containing the number of distinct values in the data structure.

The query operation can be done in constant time by comparing the count of the number of values to 1.

The deletion operation can also be done in constant time. The link from the deleted object to its value counter is followed and the value in the value counter is decremented. If this becomes zero, then .the global number of values counter is decremented.

The creation operation takes expected time that is linear in the size of the original array. If the size of the array is n, we allocate a $2n$-cell array to use for hashing the values. Some suitable scheme such as separate chaining will be used to resolve collisions. We process each element of the array in turn, incrementing its value cell if it exists, and creating one and incrementing the global number of values cell if it does not. Since the expected time to find the cell using hashing is constant, the expected running time of this operation is $O(n)$.

To do the creation operation in $O(n \log n)$ time, we can sort the original array and create an array of pointers to the new positions of each object; then creating the value cells requires one pass through the sorted array. .

One person found an even better solution, for which creation is linear time even in the worst case. The objects are put into a doubly linked list, and we create a new array containing pointers to their positions in this list. A count is kept of the number of adjacent pairs (in the linked list) which have different values.

The query operation can still be be done in constant time by comparing this count to 0.

The deletion operation is done by deleting the corresponding object from doubly linked list. It is then possible to update the count by comparing the deleted object with its predecessor and successor, and the successor with the predecessor.

Problem 2.

(a) This solution uses the paradigm of dynamic programming. The idea is to solve all of the subproblems in order from smallest to largest. Thus, the answers to the small subproblems are available when we try to solve any larger subproblem.

In this case, it is first useful to use this technique to find the sum of the frequencies for all possible subtrees (i.e. keys from i to j). As input we have n, the number of keys, and the array $freq[0..n]$, with $freq[i]$ equal to the frequency that the sought key is between key_i and key_{i+1}. The keys key_0 and key_{n+1} are $-\infty$ and ∞ respectively. We now define the array $sum[i, j]$ to be the sum of $freq[k]$ from i to j. Only the entries with $i \leq j$ need be computed. The entries are computed in order of increasing $j - i$; $sum[i, i]$ is easy to compute, and if $i <$ j then $sum[i, j] = sum[i, j - 1] + freq[j]$.

Armed with this array, we proceed to the main problem. Since in the pessimal tree no node has two non-null sons, the root is either the largest key or the smallest key. Define the weighted path length of a subtree to be the expected search time for that subtree times the frequency that the sought key is in that subtree. Here again we solve all the subproblems.

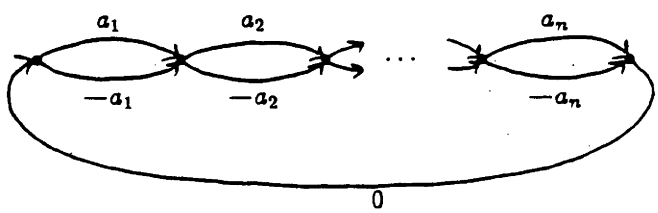
Subproblems have two indices. Subproblem [i, j] means that the sought key is between $key[i - 1]$ and $key[j + 1]$ and the given keys are $key[i]$ to $key[j]$ inclusive. Only the problems with $i < j$ are interesting. Solving a subproblem means determining weather $key[i]$ or $key[j]$ is at the root of the pessimal subtree for that problem and the weighted search time in that tree. Again the subproblems arc solved in order of increasing j $- i$ and the answers are stored in an array. If j $-$ i = 1 then there are only two possibie trees and the answer can be found quickly by exhaustion. If j $-$ 1 > 1, then the pessimal tree with $key[j]$ at the root

has weighted path length equal to the weighted path length of the pessimal subtree for subproblem $[i, j-1]$ plus the frequency that the sought key is between $key[i-1]$ and $key[j+1]$. Similarly, the weighted path length for the tree with $key[i]$ at the root the weighted path length for subproblem $[i+1, j]$ plus the same frequency. Since one of these is the pessimal tree, a comparison will yield the answer to this subproblem.

After we have finished solving problem $[1, n]$, the tree can be recovered. The root is known. The root of the rest of the tree is known. This can be iterated to cause the tree to be returned in any convenient form. The above discussion shows that the algorithm works.

(b) The algorithm takes $O(n^2)$ time. For both the summing and solving the main problem there are $O(n^2)$ subproblems that must be solved. Each subproblem takes constant time to be solved. Unwinding takes only O(n) time. Therefore the total time is $O(n^2)$.

Problem 3. The problem is clearly in NP, since a non-deterministic machine can find a cycle and verify that its edge-weights add to zero, in time which is polynomial in the number of nodes in the graph. To show that it is NP-complete, we can reduce the partition problem to it as follows: Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers, there is a set $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \sum_{a_i \in A - A'} a_i$ if and only if the graph



has a zero-cycle. (If you don't like multiple edges between a pair of vertices, you can add extra vertices to the graph above to get the same effect.)

This is because every cycle must pass through all the vertices in the graph above. Therefore a cycle is a zero-cycle if and only if the sum of the weights of the positive weighted paths is equal to the absolute value of the sum of the weights of the negative weighted paths. This defines a partition of the set A, and conversely any partition of A defines a cycle in the graph. This reduction can be carried out in time which is polynomial in n.

Another solution is to start with the problem of finding a directed Hamiltonian cycle, which is known to be NP-complete. Given a directed graph, to see if there is a Hamiltonian cycle, first label all of the edges with weight 1. Then choose any vertex v, and label all of its incoming edges with $-(n-1)$, where n is the number of nodes in the graph. If there is a zero-cycle, it must contain one of these edges, and hence must also pass through $n-1$ of the edges labeled 1. Because of the restriction that no node appears twice on a zero-cycle, this is a Hamiltonian cycle in the original graph. Conversely, if the graph has a Hamiltonian cycle, that cycle will include an edge leading into v and therefore be a zero-cycle. Therefore, the weighted graph will have a zero-cycle if and only if the original graph had a Hamiltonian cycle. This reduction can clearly be done in time which is polynomial in n, which proves that the zero-cycle problem is NP-complete.

ARTIFICIAL INTELLIGENCE

Problem 1. [Line labelling] The techniques work on the basis of a number of assumptions, such as the thoroughness of the line finder (no missing or additional lines or intersections). This would be impossible to achieve for the kinds of objects in the pictures. They are designed to recognize three-dimensional objects with special properties (e.g. faces are flat and every vertex is a junction of at most three edges) which are not true of many natural objects (such as airplanes). Although they would work for flat projections (which is what you mostly get from the air) they are not especially useful in that case.

Problem 2. [ATN's] A straightforward context-free grammar cannot deal with natural language phenomena such as agreement, and cannot be used to provide a semantically appropriate analysis for cases of "movement" like 'Which dog did you say the cat bit?". By having registers that that can be set and read (and passed up and down), ATN's can handle these phenomena. It is interesting to note that there is current work on extending the idea of context free grammars (via meta-rules of various types) to overcome the difficulties.

Problem 3. [Water witches]

(a) Production rules:

1. If wiggles and grass then stream.
2. If twirls and twitchy then stone.
3. If sand and stone then lake.
4. If jump and stone then lake.
5. If grass then not lake.
6. If wiggle and twirl then sand.

Situation (1) Grass and wiggle:
(Looking for stream (1))
Is the rod wiggling? - Yes
Are you standing on grass? - Yes

At this point stream is established, but both may be present. If the program is smart it will recognize that rule 5 can already be used, and will say 'You are over a stream." If it is not, it will take the rules in order trying to establish a lake.

(Looking for lake (3))
(Looking for sand (6))
(Rod is wiggling (already established) (6))
Is the rod twirling? - No
(Looking for lake (4))
Is the rod jumping? - No
(Looking for lake (5))
(Grass is already established)

so lake is eliminated, all search is done and answer is 'You are standing over a stream."

Situation (2) No grass, wiggle, twirl and twitch:
(Looking for stream (1))
Is the rod wiggling? - Yes
Are you standing on grass? - No
(Looking for lake (3))
(Looking for sand (6))
(Rod is wiggling (already established))

Is the rod twirling? – Yes

so it is established that there is sand . . .
(Looking for stone (2))
(Rod is twirling (already established))
Is the rod twitching? – Yes

so it is established that there is stone. Answer: "You are standing above a lake."

(b) Planning:

Actions:
Anneal
Preconditions: none
Delete: Soft, n gnarls
Add: hard, 0 gnarls
Transmogrify 1
Preconditions: soft, at least one gnarl
Delete: n gnarls, m branches
Add: $n-1$ gnarls, $m+1$ branches
Transmogrify 2
Preconditions: hard,, at least one branch
Delete: n branches
Add: $n-1$ branches
Clone
Preconditions: none
Delete: n gnarls, hard
Add: $2n$ gnarls, soft

We have separated out the action of transmogrifying into the two cases depending on hardness. We also ignore the possibility of applying the operator in cases where it will do nothing.

Initial conditions $(2, 1, hard)$ [gnarls, branches, hardness].

The plan is:

$(2, 1,$ hard) 3 $(2, 0,$ hard) [transmogrify2]
$\Rightarrow (4, 0,$ soft) [clone]
$\Rightarrow (3, 1, soft)$ [transmogrifyl]
$\Rightarrow (2, 2, soft)$ [transmogrify1]

A simple difference driven search (à la GPS) would not get an answer to this problem, since the solution calls for making apparently backward movement.

Goal $(2, 2, ?)$ – have $(2, 1, hard)$.

Difference = branches; need to add.
Try Transmogrify 1 — preconditions: needs to be soft.
Try Clone — no preconditions $\Rightarrow (4, 1,$ soft).
applying Transmogrify1 $\Rightarrow (3, 2, soft)$.

Difference = gnarls; need to remove.

Try Anneal — precondition none $\Rightarrow (0, 2, hard)$.

Difference = gnarls; need to add.

Failure — there is no way to add gnarls if there are none.

The plan would eventually be found by an exhaustive backtracking or breadth-first search.

(c) Proof:

Axioms (using arithmetic in the obvious way):

Since we are always dealing with one rod, we can treat the operations as one-argument functions from situations to situations. Similarly, *hard* and soft can be predicates associated with situations. It wouldn't harm anything to carry along a variable x standing for the rod, but it isn't of any use either.

Predicates:

$$hard(s), \quad soft(s), \quad gnarls(n, s), \quad branches(n, a).$$

Operators (functions from situation to situation):

$$anneal, \quad transmogrify, \quad clone.$$

Axioms:

$\forall s\ soft(s) \vee hard(s)$
$\forall s\ \neg(soft(s) \wedge hard(a))$
$\forall s\ hard(anneal(s))$
$\forall s\ gnarls(0, anneal(s))$
$\forall n, s\ branches(n, s) \supset branches(n, anneal(s))$
$\forall soft(a) \supset aoft(transmogrify(s))$
$\forall hard(a) \supset hard(transmogrify(a))$
$\forall s, n, k\ soft(s) \wedge gnarls(n, s) \wedge n > 0 \wedge branches(k, a)$
$\supset gnarls(n - 1, transmogrify(a)) \wedge branches(k + 1, transmogrify(a))$
$\forall k\ soft(s) \wedge gnarls(0, s) \wedge branches(k, s)$
$\supset gnarls(0, transmogrify(s)) \wedge branches(k, transmogrify(s))$
$\forall s, n, k\ hard(s) \wedge branches(0, s) \wedge gnarls(k, a)$
$\supset branches(0, transmogrify(a)) \wedge gnarls(k, transmogrify(s))$
$\forall s, n, k\ hard(s) \wedge branches(n, s) \wedge n > 0 \wedge gnarls(k, s)$
$\supset branches(n - 1, transmogrify(s)) \wedge gnarls(k, transmogrify(s))$
$\forall aoft(clone(s))$
$\forall s, n\ gnarls(n, s) \supset gnarls(2n, clone(s))$
$\forall s, n\ branches(n, s) \supset branches(n, clone(a))$

To prove: you cannot produce (1, 1, ?) from $(1, 2, soft)$. Informally: A soft rod can never decrease its branches without losing all its gnarls forever after.

Proof: To decrease branches, it must be hard and then transmogrified.
To become hard it must be annealed.
If it is annealed it loses all its gnarls.
There is no way to add gnarls to a rod that has none.

Axioms and proof using stage numbers:

Predicates:

$hard(a)$
$soft(s)$
$gnarls(n, 3)$
$branches(n, a)$
$anneal(3)$
$transmogrify(s)$
$clone(s)$

where the last three are interpreted as meaning that the named operation was applied to stage $s-1$ in order to produce stage a. Take stage 0 as the initial state.

[1] $\forall s$ a $> 0 \supset (anneal(s) \vee transmogrify(s) \vee clone(s))$
— every stage after the initial one comes from some operation —
[2] $\forall s\ anneal(s) \supset \neg(transmogrify(s) \vee clone(s))$
[3] $\forall s\ transmogrify(s) \supset \neg clone(s)$
— only one operation per stage —
[4] $\forall s\ \exists m, n\ m \geq 0\ A\ n \geq 0$ A $branches(m, a)$ A $gnarls(n, a)$
— there exist some number of branches and gnarls at each stage —
[5] $\forall s\ soft(s) \vee hard(s)$
[6] $\forall s\ \neg(soft(s)$ A $hard(s))$
[7] $\forall s\ anneal(s) \supset hard(s)$
[8] $\forall s\ anneal(s) \supset gnarls(0, a)$
[9] $\forall s, n\ branches(n, a)\ A\ anneal(s+1) \supset branches(n, s+1)$
[10] $\forall s\ soft(s)\ A\ transmogrify(s+1) \supset soft(s+1)$
[11] $\forall s\ hard(s) \wedge transmogrify(s+1) \supset hard(s+1)$
[12] $\forall s, n, k\ soft(s)\ A\ gnarls(n, a)\ A\ n > 0\ A\ branches(k, a)\ A\ transmogrify(s+1)$
$\supset gnarls(n-1, s+1)$ A $branches(k+1, s+1)$
[13] $\forall s, n, k\ soft(s)\ A\ gnarls(0, a)\ A\ branches(k, a) \wedge transmogrify(s+1)$
$\supset gnarls(0, s+1)\ A\ branches(k, s+1)$
[14] $\forall s, n, k\ hard(s)\ A\ branches(n, a)\ A\ n > 0\ A\ gnarls(k, a) \wedge transmogrify(s+1)$
$\supset branches(n-1, s+1)$ A $gnarls(k, s+1)$
[15] $\forall s, n, k\ hard(s) \wedge branches(0, a) \wedge gnarls(k, a)\ A\ transmogrify(s+1)$
$\supset \wedge branches(0, s+1)$ A $gnarls(k, s+1)$
[16] $\forall s\ clone(s) \supset soft(s)$
[17] $\forall s, n, k\ branches(k, a) \wedge clone(s+1) \supset branches(k, s+1)$
[18] $\forall s, n, k\ gnarls(n, a)\ A\ clone(s+1) \supset gnarls(2n, s+1)$

Given $branches(2, 0)\ A\ gnarls(1, 0)$ A $soft(O)$, we want to show that $\neg\exists s\ branches(1, a)$ A $gnarls(1, a)$. Proof by contradiction: assume the Conclusion.

Let x be any integer $0 < x \leq s$ such that $branches(m, x-1)$ and $branches(n, x)$, where $m > n$. Such a stage must exist since $branches(2, 0)$ and $branches(1, a)$.

$$anneal(x) \vee transmogrify(z) \vee clone(x). \qquad [1]$$

Taking cases:

$anneal(x) \wedge branches(m, x-1) \supset branches(m, x)$ [9] — contradiction,
$clone(x)\ A\ branches(m, x-1) \supset branches(m, x)$ [17] — contradiction.

So $transmogrify(x)$.

Lemma 1: $\neg soft(x-1)$. Proof by contradiction: assume $soft(x-1)$.

$$\exists j\ gnarls(j, x-1), \qquad [4]$$

and $j = 0$ or $j > 0$. Assuming $soft(x-1)$ A $j = 0$,

$soft(x-1)$ A $gnarls(0, x-1) \wedge branches(m, x-1) \supset branches(m, x)$ [13] — contradiction.

Assuming $aoft(x-1) \wedge j > 0$,

$$soft(x-1)\ A\ gnarls(j, x-1) \wedge branches(m, x-1) \supset branches(m+1, x), \qquad [12]$$

which is a contradiction since ***branches***$(n,$ x) and m > n. Lemma proved.

$$soft(x-1) \vee hard(x-1). \quad [5]$$

Therefore $hard(x-1)$, by Lemma 1. Since we have $soft(O)$ and $hard(x-1)$, there must exist $0 < y < x$ such that $hard(y)$ and $soft(y-1)$.

$$anneal(y) \vee transmogrify(y) \vee \text{clone(y)}. \quad [1]$$

Taking cases:

$$clone(y) \supset soft(y) \quad [16] \text{ — contradiction,}$$
$$aoft(y-1) \wedge transmogrify(y) \supset soft(y) \quad [10] \text{ — contradiction.}$$

So ***anneal***(y). Therefore

$$gnarls(0, y). \quad [8]$$

Lemma 2: $gnarls(0, z)$ for all $z >$ y. Proof by induction.

Induction step: Vk $gnarls(0, k) \supset gnarls(0, k+1)$.

$$anneal(k+1) \vee transmogrify(k+1) \vee clone(k+1). \quad [1]$$

Taking cases:

$$anneal(k+1) \supset gnarls(0, k+1), \quad [8]$$
$$clone(k+1) \wedge gnarls(0, k) \supset gnarls(0, k+1), \quad [18]$$

and for the case $transmogrify(k+1)$, either $hard(k)$ or $soft(k)$ by [5], so

$$soft(k) \wedge transmogrify(k+1) \wedge gnarls(0, k) \supset gnarls(0, k), \quad [13]$$

if $soft(k)$, otherwise $\exists j\ branches(j, k)$ by [4], and j = 0 or j > 0:

$$hard(k) \wedge transmogrify(k+1) \wedge gnarls(0, k) \wedge branches(0, k) \supset gnarls(0, k), \quad [15]$$
$$hard(k) \wedge transmogrify(k+1) \wedge gnarls(0, k) \wedge branches(j, k) \wedge j > 0 \supset gnarls(0, k), \quad [14]$$

so the inductive step is proved.

Base of induction: $gnarls(0, y)$ by assumption. End of Lemma.

Therefore $gnarls(0,$ a) since $s >$ x > y [Lemma 2], but this contradicts the initial assumption that $gnarls(1, s)$. Q. E. D.

ARTIFICIAL INTELLIGENCE

Problem 1. [10 points]. What major difficulties would you expect in applying line-labelling techniques (Waltz, Huffman, Clewes, etc.) to the problem of analyzing aerial photographs to detect roads, airports, missile launchers, etc.?

Problem 2. [5 points]. Why are ATN parsers better than ordinary context-free grammars for natural language understanding?

Problem 3. [three parts, 15 points each]. You have been hired as a consultant by Acme Dowsing International to help them apply AI to improve the profitability of their water exploration teams. They have asked your help in several ways.

(a) [15 points] Acme's expert dowsers have over the years built up a set of' rules of thumb for deciding what is causing the rod to dip. Somebody tried to get them to write down their rules, and produced the following:

- If the rod wiggles and you are standing on a patch of grass, then there is an underground stream below.
- If the rod twirls and is twitchy then you are over a buried stone.
- If either you are above a sand formation or the rod jumps, and also you are over a buried stone, then there is a lake below.
- There is never a lake below a patch of grass.
- Whenever the rod wiggles and twirls you are above a sand formation.

Acme wants to replace its expensive experts with a computer program that will tell what kind of body of water is causing the dip. Put these rules into a production rule form (of the kind used by MYCIN, but without certainty factors). Show a dialog that would be produced by a straightfor- ● ard backward chaining diagnosis program, for each of the following situations. Its result should be something like "'You are over a lake" or 'You are over a stream". The dialog will include questions asked of a semi-skilled dowser's helper who manipulates the rod and can answer questions like "Arc you standing on a patch of grass?" and "Is the rod wiggling?" but who has no idea about what is underground.

Situation (1): The helper is standing on a patch of grass and the rod wiggles.

Situation (2): The helper is not standing on a patch of grass and the rod wiggles, twirls, and twitches.

Note: The words used in this problem are intended to be taken as purely formal, and no conclusions should be made on the basis of their ordinary meanings. For example there is no relationship between "wiggling" and "twitching" and "jumping".

(b) [15 points] The dowsers have also discovered over the years that in certain situations different rods work best, depending on the number of branches and gnarls they have. In the old days they combed the forest for appropriate rods. Later they learned that there were alchemical methods for modifying rods, and that they could start with one that wasn't right and get the one they wanted. Only the little old rodmakers knew the secrets of producing good rods. The company wants to decrease its dependence on these rather unpredictable and sassy fellows by automating the rodmaking process. They have analyzed what the rodmakers are doing and have found that there are 3 processes obeying the following rules:

- If a rod is annealed it loses all its gnarls and becomes (or remain;) hard.
- If a rod is *transmogrified,* then if it is soft one of its gnarls becomes a branch, otherwise one of its branches falls off.
- If a rod is cloned, then the number of gnarls is doubled and it becomes (or iemains) soft.
- Whenever an operation would call for removing a branch or gnarl and there are none, the operation has no effect on the rod at all.
- Anything not mentioned above is assumed to be unchanged by the operation (e.g. annealing does not change the number of branches).

Your job is to create a knowledge base for a STRIPS-like robot planning system which can be used to generate a sequence of operations to be carried out given a raw rod and a desired form for the finished product. It should be done in a general enough way that data describing new rod-modifying processes can be added without reprogramming.

Show a plan for generating a 2 gnarl, 2 branch rod (any hardness) from a soft, 2 gnarl, 1 branch one. Show a trace of what would happen if you tried to generate it with a difference driven planning system (like GPS).

(c) [15 points] One of the little old rodmakers has argued for years that he cannot produce a 1 gnarl, 1 branch rod from a 1 gnarl, 2 branch rod using any combination of the known operations. The boss has had him keep trying, in hopes he is wrong. Represent the operations in predicate calculus using a situation variable and outline a proof that it is impossible. You do not have to give the proof in detail.

HARDWARE SYSTEMS

Problem 1. e.

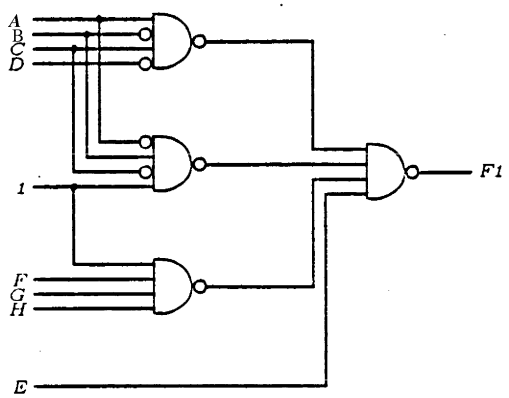
Problem 2. (a) F (b) F (c) T (d) F (e) F (f) T

Problem 3. 3. Memory addresses require only 15 bits, so all can have a 0 in the most significant bit. So let' IO addresses be distinguished by having a 1 in the most significant bit. This leaves 15 bits to encode 100 IO port addresses: any 2-out-of-15 code does the job. Each address decoder then has to test the most significant bit, and the appropriate two of the remaining bits.

Problem 4. (a) T (b) F (c) F (d) T

Problem 5. (a) 4 (b) 6 (c) 2 (d) 7 (e) 8 (f) 1, 2 (g) 1, 2, 3, 4 (h) 5 (i) 1 (j) 5

Problem 6. (a)



(b) 256 × 4.

(c) 10 AND gates: $AB'CD'$, *A'BC'*, *FGH*, *E'*, *BCD*, *CDE'H*, *FGH'*, $AB'CD$, *ABEF*, *A'BC'*, $AB'CD'FGH'$. *(F G* is implemented as *FGH + FGH'.)*

16 inputs per AND gate (each variable and its complement).

4 OR gates (1 for each function).

Problem 7. In horizontal microprogramming, each gate is directly controlled by a singie, separate bit in the microinstruction. In vertical microprogramming, functions are encoded in one or more fields; each field is decoded, and the ouput from the decoders goes to the gates. Horizontal microprogramming allows greater parallelism, but takes more space.

Problem 8. The microprogram control unit uses overlap, so that the next microinstruction is already being fetched while the current one is being executed. Since the test outcome is not known until the end of the current microinstruction, the branch must be delayed one microinstruction cycle.

Problem 1. [10 points]. Several fundamental-mode state tables are shown below. Such tables are used to describe the operation of sequential circuits built from cross-coupled gates or unclocked flip-flops. In particular, one of these state tables describes the operation of a positive-edge-triggered D flip-flop. Which one?

Hint: How to read fundamental-mode state tables: Parenthesized entries indicate stable states. In table (c) below, suppose that the circuit is in state A with input 01 (i.e. $CLK = 0$ and $D = 1$). Then if the input changes to 11 and then 10, the circuit will traverse the states shown by the arrows.

(a)

S	00	01	11	10	Q
A	(A)	B	B	(A)	0
B	(B)	A	A	(B)	1

(b)

S	00	01	11	10	Q
A	B	(A)	(A)	B	0
B	A	(B)	(B)	A	1

(c)

S	00	01	11	10	Q
A	(A)	(A)	B	(A)	0
B	(B)	(B)	(B)	A	1

(d)

S	00	01	11	10	Q
A	(A)	B	(A)	(A)	0
B	A	(B)	(B)	(B)	1

(e)

S	00	01	11	10	Q
A	(A)	(A)	C	B	0
B	A	A	(B)	(B)	0
c	D	D	(C)	(C)	1
D	(D)	(D)	C	B	1

(f)

S	00	01	11	10	Q
A	B	C	(A)	(A)	0
B	(B)	(B)	A	A	0
c	(C)	(C)	D	D	1
D	B	C	(D)	(D)	1

(g)

S	00	01	11	10	Q
A	(A)	(A)	C	C	0
B	A	A	(B)	(B)	0
C	D	D	(C)	(C)	1
D	(D)	(D)	B	B	1

(h)

S	00	01	11	10	Q
A	(A)	(A)	C	B	0
B	A	A	D	(B)	0
c	D	D	(C)	A	1
D	(D)	(D)	C	B	1

Problem 2. [6 points]. In a memory-mapped I/O system, input/output ports are addressed as memory locations and may be accessed by any memory-reference instruction. Computers that use memory-mapped I/O include the PDP-11, VAX-11, 6809, and 68000. In an isolated I/O system, input/output ports have their own address space and may be only by accessed by special I/O instructions. Computers with isolated I/O include the PDP-8, HP21MX, 8080, 8086, 280, 28000, and MCS-48. Answer TRUE or FALSE to each of the following questions.

(a) With isolated I/O, separate buses must be provided for the memory system and for the I/O system.

(b) With memory-mapped I/O, memory locations and I/O ports must have the same maximum access time.

(c) With a processor designed for isolated I/O, memory-mapped I/O may be used at the discretion of the system hardware designer.

(d) With a processor designed for memory-mapped I/O, isolated I/O may be used at the discretion of the system hardware designer.

(e) Vectored interrupts cannot be provided in a system with isolated I/O.

(f) I/O port addresses in a memory-mapped I/O system may be assigned in disjoint segments of the memory address space.

Problem 3. [4 points]. A particular computer system with 32K bytes of main memory and 100 I/O ports uses memory-mapped I/O. If the address bus contains 16 lines, what is the minimum number of AND-gate inputs that each I/O interface needs to decode its address? Explain your answer briefly. (Hint: use "m-out-of-n" codes.)

Problem 4. [8 points]. Two n-bit operands A and B are to be combined by two's-complement addition; the bits of each are numbered from 0 (least significant bit) to n — 1 (sign bit). Let S denote the sum; let C(n — 1) denote the carry from bit n — 2 into bit n — 1 when A and B are added; and let C(n) denote the carry out of bit n — 1. Indicate whether each of the following conditions is a valid test for two's-complement overflow. (The condition must be true if and only if there is overflow.) Answer TRUE or FALSE.

(a) $C(n) \neq C(n-1)$.

(b) $A(n-1) \oplus B(n-1) = C(n)$.

(c) $A(n-1) \oplus B(n-1) \neq C(n)$.

(d) $A(n-1) \oplus B(n-1) = 0$ and $A(n-1) \neq S(n-1)$.

Problem 5. [10 points]. Memory hierarchies. Several different types of computer memory are listed below, followed by certain memory characteristics.

(1) Semiconductor RAM (main memory)
(2) Semiconductor RAM (cache memory)
(3) Erasable Programmable Read-Only Memory (EPROM)
(4) Core memory
(5) Floppy disk
(6) Moving-head (Winchester) disk
(7) Head-per-track disk
(8) None of the above

For each characteristic below, list the memory type(s) above which have that characteristic.

(a) Erased by every read operation

(b) Lowest cost per bit

(c) Highest cost per bit

(d) Best for external storage in demand-paging systems

(e) First type of memory technology used in computers

(f) Volatile

(g) Random access capability

(h) Longest access latency

(i) Highest storage density (bits per unit area) in storage medium.

(j) Highest storage density (bits per unit volume) in computer room.

Problem 6. [12 points]. The following four boolean equations describe a 4-output logic function. Apostrophes (') denote compiementation.

$$F1 = AB'CD' + A'BC' + FGH + E'$$
$$F2 = BCD + CDE'H + FGH'$$
$$F3 = AB'CD + ABEF + FG$$
$$F4 = A'BC' + AB'CD'FGH'$$

(a) [5 points] Draw a circuit diagram for $F1$ using only 4-input NAND gates.

(b) [3 points] If the 4-output function is implemented with a read-only memory, what size ROM is needed?

(c) [4 points] If the 4-output function is implemented with a programmable logic array, describe the organization of the PLA by giving:

- [2 points] The number of AND gates (also list the corresponding AND terms from the equations above for each one),
- [1 point] The number of inputs per AND gate,
- [1 point] The number of OR gates.

Problem 7. [5 points]. Explain the difference between vertical and horizontal microprogramming.

Problem 8. [5 points]. In some microprogrammed processors, conditional microprogram branches take place one microinstruction after the branch microinstruction is executed. For example, the following microcode implements the machine instruction DJNZ R ,ADDR (Decrement and Jump if Not Zero).

Microprogram Address	Instruction
.	.
.	.
5	Load R
6	Decrement R
7	Branch to 11 if R = 0
8	Store R
9	Load ADDR
10	Store PC
11	Fetch next macro instruction
.	.
.	.

In this example, instruction 8 is executed whether the branch is taken or not. Instructions 9 and 10 are executed only if the branch is not taken.

Now the question: Why is the microprogrammed processor designed this way?

Problem 1. Let $A^{(1)} = A$. Then the algorithm for computing the Cholesky factorization goes as follows:

$$
\begin{aligned}
&\text{For } k = 1, 2, \ldots, n \\
&\quad f_{ik} = 0, \qquad i = 1, 2, \ldots, k-1 \\
&\quad f_{kk} = (a_{kk}^{(k)})^{1/2} \\
&\quad f_{ik} = a_{ik}/f_{kk}, \qquad i = k+1, \ldots, n \\
&\quad a_{ij}^{(k+1)} = a_{ij}^{(k)} - f_{ik}f_{jk}, \qquad j \geq i = k+1, \ldots, n.
\end{aligned}
$$

Thus n square roots and $n^3/6 + O(n^2)$ flops (a combination of a floating multiply followed by a floating add) are required for computing the Cholesky factor.

(a) If the matrix is **5-diagonal,** then

$$f_{ik} = 0 \qquad \text{when } i < k \text{ or } i > k + 2.$$

The above equations show that n square roots, and about $2n$ divisions and $3n$ flops are required for computing the Cholesky factor.

F has only three non-zero diagonals;

$$F = \begin{pmatrix} & & & 0 \\ & & & \\ & & & \\ 0 & & & \end{pmatrix}$$

and thus requires at most $3n - 3$ words.

(b) The matrix A appears thusly:

$$A = \begin{pmatrix} & & & 0 \\ & & & \\ & & & \\ 0 & & & \end{pmatrix}.$$

After one step of the Cholesky factorization, the zeros are destroyed, so that $n^3/6 + O(n^2)$ flops are required for computing the factorization.

(c) We can simplify the problem by permuting $A((n, n-1, \ldots, 1) \leftarrow (1, 2, \ldots, n))$ so that

$$PAP^T = \begin{pmatrix} & & 0 \\ & \iddots & \\ 0 & & \end{pmatrix}.$$

Now F requires $2n-1$ words; and n square roots, $n-1$ divisions, and $n-1$ flops are needed to' compute the factorization.

Problem 2.

(a) To apply the Aitken procedure for estimating β, we must eliminate k in the βk term. This can be done by noting that

$$x_{k+1} - x_k = \beta + \sum_{k=1}^{\infty} \tilde{\alpha}_r \lambda_r^k,$$

where $\tilde{\alpha}_r = \alpha_r(\lambda_r - 1)$. Then if we apply the Aitken scheme to

$$y_k = x_{k+1} - x_k,$$

the extrapolated value will approximate β.

(b) If

$$y_k = \beta + \tilde{\alpha}_1 \lambda_1^k,$$

then from y_0, y_1, y_2 we can determine β since we have three unknowns $\beta, \tilde{\alpha}_1, \lambda_1$. Hence if

$$x_k = \alpha + \beta k + \alpha_1 \lambda_1^k,$$

we can determine β precisely.

Problem 3.

(a) Note that

$$x_{n+1}^* - \alpha = f(x_n^*, y_n^*) - f(\alpha, \beta) = \frac{\partial f}{\partial x}(x_n^* - \alpha) + \frac{\partial f}{\partial y}(y_n^* - \beta) + \cdots,$$
$$y_{n+1}^* - \beta = g(x_{n+1}^*, y_n^*) - g(\alpha, \beta) = \frac{\partial g}{\partial x}(x_{n+1}^* - \alpha) + \frac{\partial g}{\partial y}(y_n^* - \beta) + \cdots$$

Hence

$$\begin{pmatrix} 1 & 0 \\ -\frac{\partial g}{\partial x} & 1 \end{pmatrix} \begin{pmatrix} x_{n+1}^* - \alpha \\ y_{n+1}^* - \beta \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ 0 & \frac{\partial g}{\partial y} \end{pmatrix} \begin{pmatrix} x_n^* - \alpha \\ y_n^* - \beta \end{pmatrix} + \cdots.$$

Since

$$\begin{pmatrix} 1 & 0 \\ -a & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix},$$

$$J^*(\alpha, \beta) = \begin{pmatrix} 1 & 0 \\ \frac{\partial g}{\partial x} & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ 0 & \frac{\partial g}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} \cdot \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \cdot \frac{\partial g}{\partial x} + \frac{\partial g}{\partial y} \end{pmatrix}.$$

(b) We wish to show

$$||J^*(\alpha,\beta)|| \leq ||J(\alpha,\beta)||$$

when

$$||J(\alpha,\beta)|| < 1.$$

Since the first rows of $J(\alpha,\beta)$ and $J^*(\text{a},\beta)$ are the same, we simply need to check the second row of $J^*(\alpha,\beta)$. Now

$$\begin{aligned}\left|\frac{\partial g}{\partial x}\cdot\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\cdot\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}\right| &\leq \left|\frac{\partial g}{\partial x}\right|\cdot\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\cdot\left|\frac{\partial g}{\partial x}\right| + \left|\frac{\partial g}{\partial y}\right| \\ &\leq \left|\frac{\partial g}{\partial x}\right|\left(\left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right|\right) + \left|\frac{\partial g}{\partial y}\right| \\ &\leq \left|\frac{\partial g}{\partial x}\right| + \left|\frac{\partial g}{\partial y}\right|.\end{aligned}$$

The last step follows since $||J(\alpha,\beta)|| < 1$. Thus $||J^*(\alpha,\beta)|| \leq ||J(\alpha,\beta)||$ when $||J(\alpha,\beta)|| < 1$.

NUMERICAL ANALYSIS

Problem 1. Linear systems [20 points]. Let A be a real, symmetric, positive definite matrix. In this problem, we assume the matrix A is sparse and we wish to investigate means of taking advantage of the structure. We desire to compute the Cholesky factor F of A so that $FF^T = A$. Note that since A is positive definite it is not necesssary to pivot for numerical stability.

(a) [9 points] Assume that $a_{ij} = 0$ when $|j - i| \geq 3$, i.e. A is a five diagonal matrix. Describe an efficient variant of the Cholesky method for finding F. How much storage does F require? How many numerical operations does your algorithm require?

(b) [5 points] Assume that the first row and column of A are non-zero but that $a_{ij} = 0$ when $i \geq 2$, $j \geq 2$, and $i \neq j$. How many operations are required to find the factor F and how much storage is required?

(c) [6 points] Show how to reorder the matrix given in (b) so that the storage and the operations are reduced. Give a count of the number of operations after this improvement.

Problem 2. Acceleration [15 points]. Consider a sequence $x_k, k = 0, 1, \ldots$ which satisfies the relationship

$$x_k = \alpha + \beta k + \sum_{r=1}^{\infty} a_r \lambda_r^k,$$

where α, β, and $\{a_r, \lambda_r\}_{r=1}^{\infty}$ are unknown constants with $|\lambda_r| < 1$ and $|\lambda_r| > |\lambda_{r+1}|$.

(a) [8 points] Given numerical values x_0, x_1, x_2, x_3, show how to use Aitken acceleration to determine an approximate value of β.

(b) [7 points] Under what circumstances will your algorithm yield the exact value of β when x_0, xi, x_2, and x_3 are given?

Problem 3. Non-linear equations [25 points]. We wish to solve the system

$$x = f(x, y),$$
$$Y = g(x, y).$$

Let us assume that a solution exists, which we denote as x = α, y = β. Consider the following iteration schemes:

I	II
$x_{n+1} = f(x_n, y_n)$	$x^*_{n+1} = f(x^*_n, y^*_n)$
$y_{n+1} = g(x_n, Yn)$	$y^*_{n+1} = g(x^*_{n+1}, y^*_n)$

with $x_0 = x^*_0, y_0 = y^*_0$. Let

$$\vec{d}_n = \begin{pmatrix} x_n - \alpha \\ y_n - \beta \end{pmatrix}, \qquad \vec{d}^*_n = \begin{pmatrix} x^*_n - \alpha \\ y^*_n - \beta \end{pmatrix},$$

so that

$$\vec{d}_{n+1} = J(\alpha, \beta)\, \vec{d}_n + \vec{\delta}_n,$$

where J is the Jacobian associated with f and g.

(a) (15 points] Show that

$$\vec{d}^*_{n+1} = J^*(\alpha, \beta)\, \vec{d}^*_n + \vec{\delta}^*_n.$$

Give an expression for J^* (Q, β), assuming $\vec{\delta}^*_n$ consists of higher-order terms.

(b) [10 points] For a given matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

we define $||A||$ = max($|a| + |b|, |c| + |d|$). Show that if $||J(\alpha, \beta)|| < 1$, then $||J^*(\alpha, \beta)|| \leq ||J(\alpha, \beta)||$.

SOFTWARE SYSTEMS

Problem 1.

```
x := f 1(z);
while p1(x) do
    begin
    x := f2(x);
    if not p2(x) then
        begin
        while p3(x) do x := f3(x);
        x := f 1(z)
        end
    end
```

Problem 2.

(a) The following program assigns true to *v* if the activation environment is used, and false if the declaration environment is used.

```
begin
    boolean v, x;

    function p;
        p := x

    function q(r); function r;
        begin
            boolean x;
            x := true;
            q := r
        end;

    x := false;
    v := q(p)
end
```

(b) Block-structured languages typically use the "declaration environment". These languages are usually compiled, and greater runtime efficiency is obtained by static binding of variable references at compile-time. LISP, which is usually implemented by an interpreter, provides greater flexibility by delaying binding until run-time' and thus uses the "activation environment".

Problem 3.

(a) Static links are less efficient for resolving non-local references, because (possibly long) chains of pointers have to be followed; with displays, a single indexed, indirect addressing operation is sufficient.

Static links are more convenient and economical to maintain, especially if the langauge permits procedural/ functional parameters or call by name, both of which require context changes that are not simple pushes or pops of the activation record stack.

It can be assumed that variable references are much more frequent than the context changes which require updates of the static chain/display. This favors displays.

However, it can also be assumed that the vast majority of variable references are either local or global to the entire program. Program global variables can be stored in a separate, static area, and be accessed directly. This favors static links.

(b) Neither static links nor displays are needed; local variables are in the topmost activation record pointed to by the stack pointer, and, as mentioned in part (a), program globals can be stored in a separate, static area and be addressed directly. So the non-local accesses allowed are particularly efficient.

Problem 4.

(a) Because G1 is unambiguous, whereas G2 is ambiguous.

(b) (1) Associativity and precedences of operators (“,” and “;”).

(2) There will be fewer states in the generated parser, so generation time, parser size and parsing time will all be reduced.

Problem 5. The line numbers refer as closely as possible to those of the original program.

```
4.    s := 0 ;
7.    T := 10 + U;
9.  V := T ;
      PQ := R * 10;
5.    repeat
7.          PQ := PQ — 10;
9.          S := S + PQ
10.         until PQ ≤ 10
```

PQ is a new temporary’ holding the value that was previously *P* * Q. Optimizations:

(a) Substitution of 10 for Q everywhere: constant folding.

(b) Elimination of lines 1 and 2: dead variable (redundant store) elimination.

(c) Elimination of line 3, and replacement of *P* by *R* everywhere: copy propagation.

(d) Movement of lines 7 and 9 out of the loop: code motion.

(e) Replacement of (Q + *U*) by *T* in line 9: common sub-expression elimination.

(f) Replacement of “*” in line 9 by “—” in line 7: reduction in strength.

(g) Elimination of *P* from loop: induction variable elimination.

Problem 6.

(a) Deadlock in monitors can occur in many ways. The most obvious is having a procedure in monitor *A* call a procedure in monitor *B*, and vice versa. If one process calls the procedure in A at the same time that another calls the procedure in *B*, neither call will be able to complete because both monitors are locked.

Deadlock in a message-passing system would occur if process 1 was waiting to receive a message from process 2, which was itself waiting to receive a message from process 1.

(b) Starvation occurs when some process does not get a resource it wants; for example, because higher-priority processes monopolize the resource. Deadlock implies that process(es) are permanently blocked, and cannot recover without outside intervention. With starvation, it remains possible that the process will eventually get the resource and be able to continue.

(c) The following solution uses an array of semaphores, one per process.

```
processi: while true do
          begin
              non-critical section;
              P(sem[i]);
              critical section;
              V(sem[i + 1 mod 10])
          end;
```

The required initialization is to have every semaphore except $sem[1]$ equal 0, and $sem[1] = 1$.

Problem 7. (a) 21, 1000, 6, 6, 6. The idea here is that a long job (52) keep some short jobs (J3, J4 and J5) waiting for a long time under FCFS. T1 = 21 is to ensure that J2 . . . J5 are all already in the queue when the large job (J2) starts.

(b) 21, 100, 100, 50, any. The idea this time is that a short job (J4) keep at least two earlier, longer jobs (J2 and J3) waiting at least an extra 10 time units under SJN.

(c) 20, 15, 20, 10, 10. Short jobs get high priority; this is one way of implementing SJN. Alternatively: 1, 1, 1, 1, 1. If the queue never contains more than one job at a time, the scheduling method is immaterial!

(d) 100, 50, 100, 10, 6. The idea here is to cause at least 3 preemptions, costing an extra context switch each (note that preemption actually causes two context switches, but at least one is always required to run the job, whatever the scheduling method). The above jobstream causes 3 as shown below (* indicates preemption):

J1 → * J2 → * J4 → * J5 → J4 → J2 → J1 → J3.

SOFTWARE SYSTEMS

Problem 1. Structured Programming [8 points]. The following is an adaptation of a program extract that was actually published (!). Rewrite it in a clearer fashion, by applying the principles of structured programming.

```
A1: x := f1(x);
L1: if p1(x) then
        begin
            x := f2(x);
            if p2(x) then go to L1;
            B1: if p3(x) then
                begin
                    x := f3(x);
                    go to B1
                end
            go to A1
        end
```

Problem 2. *"Funarg"* Problem (10 points]. Consider a block-structured language (e.g. Algol60) that allows a formal parameter, say *FP,* of a function, say *F,* to be a function name. When *FP* is used in an expression within *F,* the function represented by the corresponding actual parameter, say *AP,* is invoked. There are at least three possible environments in which it could be executed:

(1) The "activation environment": the environment at the point of-call of *FP* in the expression.

(2) The "binding environment": the environment at the point of call of *F,* when *FP* was bound to *AP.*

(3) The "declaration environment": the environment at the point of declaration of the function represented by *AP.*

The choice of which environment is actually used in a particular language is a language design decision.

(a) [6 points] Write a program extract in some informal high-level notation that would assign a different value to a variable *V* depending on whether the "activation" or "declaration" environment is used.

(b) [4 points] Which of the three environments listed above is commonly used in block-structured languages (e.g. Algol60)? In LISP? Are these choices consistent with the philosophies and features of these languages? Explain briefly (5 lines maximum).

Problem 3. Static Links versus Displays [6 points]. Two alternative methods of implementing references to non-local variables in block-structured languages are "static links" (or "static chains") and "displays".

(a) [4 points] What considerations apply in choosing between these two methods?

(b) [2 points] Certain languages (e.g. BCPL and C) allow a procedure to access only local variables and variables global to the entire program (i.e. declared at the top or program level). How does this affect the implementation of non-local variable references? .

Problem 4. Parsing [6 points]. The following is a simple grammar for lists of identifiers separated by commas or semicolons:

G1: L ::= CL | L;CL
CL ::= id | CL, id

id stands for an identifier, and can be considered a terminal symbol.

An alternative grammar for the same lists is:

G2: L ::= L;L | L,L | id

(a) (2 points] An automatic parser constructor would normally prefer to deal with G1 than with G2. Why?

(b) G2 can be used as the basis for automatic parser construction, provided some additional information is provided.

(1) [2 points] What information is necessary?

(2) [2 points] What are the advantages, if any, of using G2 instead of G1 when constructing an SLR or LALR parser?

Problem 5. Optimization [10 points], Apply the standard code improvement transformations (optimizations) used by optimizing compilers to the following program segment. Show the optimized program and identify the optimizations used, stating the type (class) of each. Work entirely in the source language; do not generate code. You may assume that no two variables are aliases of each other, and that all variables except *P* and Q are live on exit. *P* and Q are dead on exit.

```
1.   P := 3;
2.   Q := P + 7;
3.   P := R;
4.   S := 0;
5.   repeat
6.        P := P - 1;
7.        T := Q + U;
8.        S := S + P * Q;
9.        V := Q + U
10.  until P ≤ 1
```

Problem 6. *Synchronization* [10 points].

(a) [2 points] Give an example of deadlock caused by monitors or message-passing.

(b) [1 point] What is starvation? How does it differ from deadlock?

(c) Suppose we have ten processes (numbered 0 through 9) which occasionally wish to have exclusive access to some resource. Any process that is not currently using the resource is allowed to work as it pleases. A process i which requires the critical resource cannot use it unless no other process is using it and process ((i — 1) mod 10) was the last process to use it.

(1) [6 points] Using semaphores (or arrays of semaphores), monitors, or messages, describe how the above synchronization may be implemented. Do not write detailed code!

(2) [1 point] What is the appropriate initialization such that when the system is restarted it appears that process 0 has just used the resource?

Problem 7. Scheduling [10 points]. Consider the following jobstream presented to a scheduler:

Job.	Arrival time	Processing time required	Priority
J1	0	T1	low
J2	5	T2	medium
J3	10	T3	low
J4	15	T4	high
J5	20	T5	high.

Arrival times and processing times required are given in the same time units (e.g. milliseconds).

Assume there is an overhead of 0.1 time units involved each time the scheduler changes the job being run. For each of the following conditions, supply integral values for the processing times T1, T2, . . . , T5, so that the condition is satisfied for the above jobstream:

(a) [3 points] First-come first-served scheduling results in at least twice as large a mean response ratio as shortest job next scheduling.

(b) [3 points] Shortest job next scheduling causes the response time of two jobs to be at least 10 time-units longer than they would be under first-come first-served scheduling.

(c) [2 points] Non-preemptive priority scheduling is equivalent to shortest job next scheduling for this jobstream.

(d) [2 points] Preemptive priority scheduling has an overhead of at least 0.3 time units more than non-preemptive priority scheduling.

Note: Use your intuition about the scheduling methods involved to arrive at suitable processing times; it should not be necessary to do detailed calculations.

Problem 1.

(a) The following assertions assume that the domain of all variables is the natural numbers; i.e. relations such as $0 \leq \kappa$ are implicit.

$\ell_2 : \forall \kappa.\ \kappa < k \supset a[\kappa] = 0$

$\ell_3 : \forall \kappa.\ \kappa \leq N \supset a[\kappa] = 0$

$\ell_4 : \forall \kappa.\ \kappa \leq \mathrm{N} \supset a[\kappa]$ = (# of pairs (i', j') such that $foo(i', j') = \kappa \wedge i' < i$)

$\ell_5 : \forall \kappa.\ \kappa \leq \mathrm{N} \supset a[\kappa]$ = (# of pairs (i', j') such that $foo(i', j') = \kappa \wedge (i' < i \vee (i' = i \wedge j' < j))$)

$\ell_6 : \forall \kappa.\ \kappa \leq \mathrm{N} \supset a[\kappa]$ = (# of pairs (i', j') such that $foo(i', j') = \kappa \wedge i' \leq i$)

$\ell_7 : \forall \kappa.\ \kappa \leq \mathrm{N} \supset a[\kappa]$ = (# of pairs (i', j') such that $foo(i', j') = \kappa$)

$\ell_8 :$ (ℓ_7 assertion) $\wedge\ \forall \kappa.\ \kappa < k \supset a[\kappa] \leq 1$

done : (ℓ_7 assertion) A $((k > N$ A $\forall \kappa.\ a[\kappa] \leq 1) \vee (k \leq N$ A $a[k] > 1 \wedge \forall \kappa.\ \kappa < k \supset a[\kappa] \leq 1))$.

The assertion at ℓ_6 is not absolutely necessary since the assertion at ℓ_4 can still be proved without it. The expression (i' $< i \vee (i' = i \wedge j' <$ j)) at ℓ_5 is a bit tricky. Another important point is that the assertion at ℓ_7 needs to be kept as part of the later assertions, since the partial correctness of the program is a statement about *foo*, not about the array a.

(b) We need to show that the loops at ℓ_2, ℓ_4, ℓ_5, and ℓ_8 all terminate. The first and the last are easy, since they can be shown to execute at most N times. The ℓ_5 loop terminates because j is incremented on each pass through the loop, while i remains constant, so by induction we can show that eventually *foo(i, j) > N*. Similarly the sequence $foo(0, 0)$, foo(l,O), $foo(2, 0), \ldots$ is strictly increasing, so eventually *foo(i, 0) > N* for some i .

Problem 2. If both A and B are regular, then so is $A \odot B$. To prove this, let $M_A = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting A, where $Q = \{q_0, \ldots, q_m\}$ is its set of states, $F \subseteq Q$ is the set of final states, and $6 : Q \times \Sigma \to Q$ is the transition function; and let M_B = (R, Σ, η, r_0, G) similarly be a DFA accepting B. We can construct a non-deterministic finite automaton $M_{A \odot B}$ which accepts A $\odot$ *B as* follows: let $M_{A \odot B} = (Q \times R, \Sigma, \gamma, (q_0, r_0), F \times G)$, where the transition function $7 : Q \times R \to 2^{Q \times R}$ is defined by

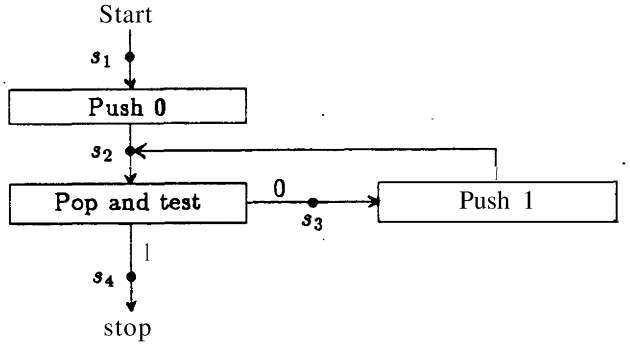
$$\gamma((q_i, r_j), a) = \{(\delta(q_i, a), r_j), (q_i, \eta(r_j, a))\} \quad \text{for } a \in \Sigma.$$

$M_{A \odot B}$ works by simulating a step from either M_A or M_B on each transition, and "remembering" the state of the other machine while doing so.

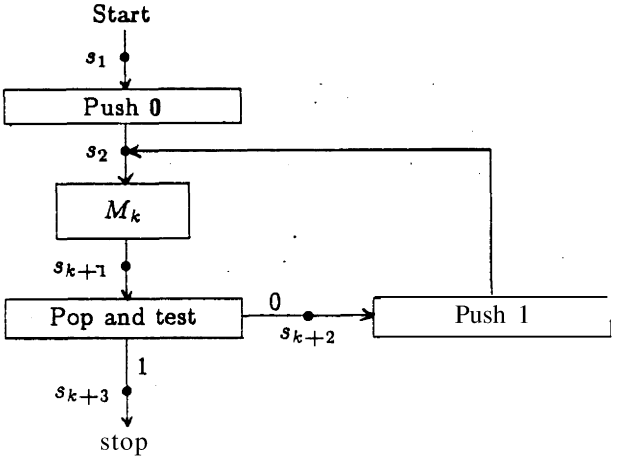
To see that this accepts $A \odot B$, first let w be a string accepted by $M_{A \odot B}$. If we examine the transitions of $M_{A \odot B}$ as it accepts w, let $a_1, \ldots,$ a, be the input symbols which cause transitions of the form $(q_i, r_j) \to (\delta(q_i, a_k), r_j)$, and let $b_1, \ldots, b_t$ be the input symbols which cause transitions of the form $(q_i, r_j) \to (q_i, \eta(r_j, a_k))$, then it is clear that $a_1 \ldots$ a, $\in A$ and $b_1 \ldots b_t \in B$. By appropriately adding null strings, we can convert $a_1 \ldots$ a, to $x_1 \ldots x_n$ and $b_1 \ldots b_t$ to $y_1 \ldots y_n$ so that w = $x_1 y_1 \ldots x_n y_n$.

Conversely, if w = $x_1 y_1 \ldots x_n y_n \in A \odot B$, then M_A accepts $x_1 \ldots$ x,, and M_B accepts $y_1 \ldots y_n$, and the sequences of transitions taken by these two machines can be used to construct a sequence of transitions of $A4_{A \odot B}$ which accepts w.

Problem 3. The basic idea is that an autonomous pushdown machine can be made to "count" in binary on its stack, and terminate when the count reaches a certain point. As an example, take $k = 4$, and let M_4 be the machine corresponding to the following diagram:



It is clear that M_4 runs for four steps and then halts with the stack restored to its initial condition. We can similarly define M_5 and Ms. Now, given M_i, we can build M_{i+3} as follows:



This machine runs for $4 + 2N_i$ steps, where N_i is the numbei of steps that M_i runs. Thus it is clear (and can be shown by induction) that $N_i > 2^{i/4}$ for $i \geq 4$. Therefore, we can let $B = 2^{1/4}$.

Problem 1. [20 points]. Let $foo(i, j)$ be a function whose arguments and value are always non-negative integers, and such that for all i and j,

$$foo(i, j) < foo(i + 1, j)$$
$$\text{and} \quad foo(i, j) < foo(i, j + 1).$$

The following program is intended to find the smallest k in the range $0 \leq k \leq N$ for which $foo(i, j) = foo(i', j') = k$ for two different pairs (i, j) and (i', j').

```
ℓ1 :  k := 0;
ℓ2 :  a[k] := 0;
      k := k + 1;
      if k ≤ N then go to ℓ2;
ℓ3 :  i := 0;
ℓ4 :  if foo(i, 0) > N then go to ℓ7;
      j := 0;
ℓ5 :  if foo(i, j) > N then go to ℓ6;
      a[foo(i, j)] := a[foo(i, j)] + 1;
      j := j + 1;
      go to ℓ5;
ℓ6 :  i := i + 1;
      go to ℓ4;
ℓ7 :  k := 0;
ℓ8 :  if k > N then go to done;
      if a[k] > 1 then go to done;
      k := k + 1;
      go to ℓ8;
done : if k > N then print(" LOSE")
           else print( "The smallest k is ", k);
```

(a) [15 points] What assertions should be attached to what labels in order that its partial correctness can be proved by the method of inductive assertions? (Give just the assertions, not the proof.)

(b) [5 points] What is involved in proving its termination? (Give an informal description; the proof itself is not required.)

Problem 2. [20 points]. Let A and B be languages over an alphabet Σ, and define the "shuffle" of A and B to be

$$A \odot B = \{w \in \Sigma^* \mid w = x_1 y_1 \ldots x_n y_n\},$$

where each x_i and each y_i is either a member of Σ or is the empty string, $x_1 \ldots x_n \in A$, and $y_1 \ldots y_n \in B$. That is, $A \odot B$ is the set of strings that we can get by "shuffling" a string from A into a string from B. If both A and B are regular sets, is $A \odot B$ regular? Give a proof of your answer.

Problem 3. [20 points]. Consider the set S of autonomous, terminating, pushdown store machines. "Autonomous" means they have no input. "Terminating" means reaching a designated "terminal" control state. "Pushdown store" means that the only memory, except for a finite number of control states, is a single pushdown stack, initially empty, over a two-character alphabet, say 0 and 1.

Show that S contains an infinite sequence of machines $\{M_i \mid i = k, k + 1, \ldots\}$ for some integer k, where each M_i has i control states and runs for at least B^i steps before terminating, for some constant $B > 1$, which is independent of i. That is, show that S contains machines whose running time is exponential in the number of control states.

Magic Number _____

Spr i ng 1980-81

Computer Science Comprehensive Exam
Written Exam
Saturday, May 9, 1981 (9:00 12:oo; 1:30 - 4:30)

READ THIS FIRST

1. The exam contains questions drawn from six areas of computer science. The total possible score is 360 points, 60 in each area. Hint: 6 hours equal 360 minutes, this may help you plan your time.

2 Please do your best to relax during the lunch break. You may not consult any references or colleagues or write drafts of answers during thi s period. Just relax.

3 Be sure that you have all 16 pages of the Exam. Your answers are to be written in blue bobks. Use a separate blue book for each of the six subject areas. Write your exam number in the upper right-hand corner of every page on which you have any solution to any problem. Please write legibly, with a pen or sharp soft pencil.

4 Strategic considerations: (a) To pass this exam at the Ph.D. level, you should not leave any of the six subject areas completely blank. as there wi 11 be a minimum competence requirement of roughly 20 points in each area. The total scores of everybody who passes this minimum requirement will then be used to determine whether or not the written exam as a whole is passed. You should plan your exam-taking strategy accordingly. (b) To pass this exam at the Masters" or CS Minor level, simply try to maximize your total score.

5. Show your work, as partia-1 credit will be given for incomplete answers.

6. This exam is open book: You may use whatever books and notes you have already brought with you and any library books provided by the committee.

7. Sign the honor code statement below and turn in this page with your 6 blue books. This page will be separated from your blue books prior to the grading process.

8. The committee suggests that you read over the entire exam quickly once, in order to help in allocating your time. We also suggest that you refrain from panic. GOOD LUCK.

9. A committee member will be available to answer questions.

in recognition of and in the spirit of the Honor Code, I certify that I have neither received nor given unpermitted aid on this exam.

Signed _____