Final Exam

CSC 173

18 December 2001

Directions; PLEASE READ

This exam has **10** questions, many of which have subparts. Each question indicates its point value. The total is 100 points. **Questions 5c and 6 are for extra credit only**; they are not part of the 100 points.

Please show your work here on the exam, in the space given. Do not write on the backs or in the margins. Put your name on every page. If your answer won't fit in the given space then you're trying to write too much. Scratch paper is available if you need it, but the proctor will collect **only the exams**.

I have tried to make the questions as clear and self-explanatory as possible. If you do not understand what a question is asking, make some reasonable assumption and *write that assumption down* next to your answer. The proctor has been instructed not to answer questions during the exam.

You will have a maximum of three hours to work. Good luck!

1. (6 points) Give three reasons why, from a programmer's point of view, it is good to work with abstract data types, rather than working directly with the data structures from which the ADTs are built.

   Abstractions serve several purposes. (1) They reduce "conceptual load", making it easier for programmers to think about and understand their code. They are, to some degree, "self-documenting". (2) By establishing narrow, well-defined interfaces, they help to isolate errors, making debugging easier, and making errors less likely in the first place. (3) By hiding knowledge of data structure details behind the interface, they localize the changes that need to be made if a data structure is changed. (4) Because interfaces can be well defined, they make it easier to reuse code in other applications. (5) On a large project they make it easier to divide work among programmers.

2. (a) (3 points) For assignment 5 you implemented Prim's minimum spanning tree algorithm for dense graphs. What is its asymptotic running time? Assume that the graph has N nodes and M edges.
   $O(N^2)$.

(b) (3 points) You will remember from lecture and from the book that Kruskal's algorithm is faster for sparse graphs. What is its asymptotic running time?

$O(M \log M) = O(M \log N)$.

(c) (6 points) Explain why Prim's algorithm is better for dense graphs, while Kruskal's algorithm is better for sparse graphs.

In a dense graph $M = O(N^2)$*, so* $O(N^2) < O(M \log N)$*; in a sparse graph* $M = O(N)$*, so* $O(N^2) > O(M \log N)$.

(d) (4 points) What data structure is used to represent the graph in Prim's algorithm? What data structure is used in Kruskal's algorithm?

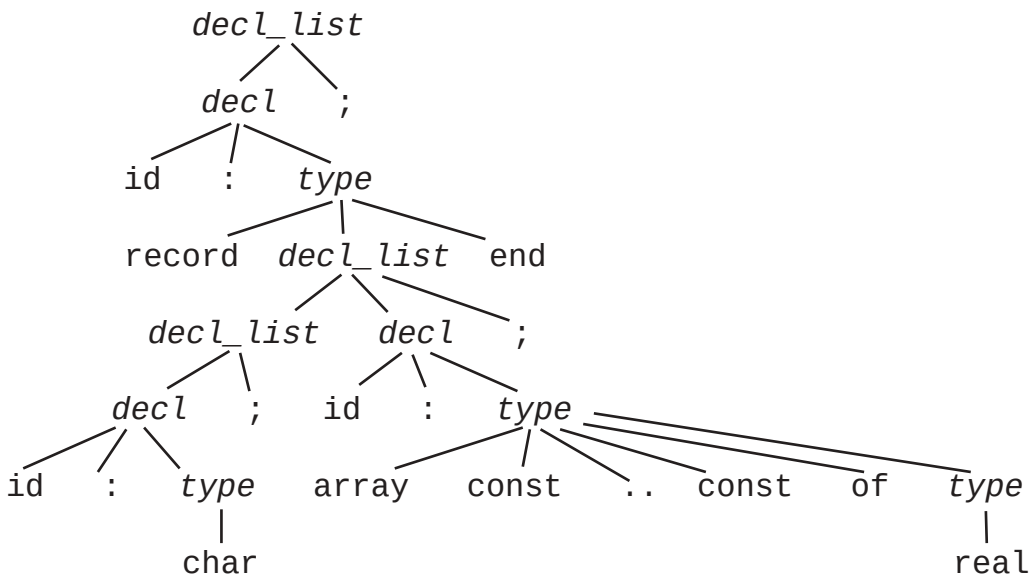
Like most dense graph algorithms, Prim's algorithm uses an adjacency matrix. Like most sparse graph algorithms, Kruskal's algorithm uses adjacency lists.

3. Consider the following CFG:

decl_list ⟶ *decl_list decl* `;`
decl_list ⟶ *decl* `;`
decl ⟶ `id :` *type*
type ⟶ `int`
type ⟶ `real`
type ⟶ `char`
type ⟶ `array const .. const of` *type*
type ⟶ `record` *decl_list* `end`

(a) (6 points) Give a parse tree for

```
foo : record
        a : char;
        b : array 1..2 of real;
    end;
```



(b) (5 points) Explain why the grammar is not LL(1) (can't be parsed with a non-backtracking recursive descent parser).

The first production for `decl_list` *is left-recursive, so we can't tell when to predict it (as opposed to predicting the second).*

(c) (4 points) Show how to modify the grammar to make it LL(1).

Replace the first two productions with

$$\textit{decl\_list} \longrightarrow \textit{decl}\ \texttt{;}\ \textit{decl\_list\_tail}$$
$$\textit{decl\_list\_tail} \longrightarrow \textit{decl}\ \texttt{;}\ \textit{decl\_list\_tail}$$
$$\textit{decl\_list\_tail} \longrightarrow \epsilon$$

4. (6 points) Describe two common types of program bugs in C that can result in a "segmentation fault" message at run time.

(1) Dereferencing an uninitialized or dangling pointer. (2) Indexing into an array with an invalid subscript (C has no array bounds checking). (3) Passing an inappropriate type of parameter to a routine such as `scanf`*, which expects pointers, but takes a variable number of arguments and hence cannot be type-checked by the compiler.*

NB: Students received partial credit for listing certain other run-time errors (e.g. infinite recursion) that fail with other kinds of messages.

5. Consider a database with the following scheme (borrowed from the Fall 2000 midterm):

Route: flight number (FN), departure city (DC), arrival city (AC), distance in miles (DI), scheduled departure time (SD), scheduled arrival time (SA), plane model (PM).

Flight: flight number (FN), date (DT), number of passengers (NP), actual departure time (AD), actual arrival time (AA).

Plane: plane model (PM), capacity (CA), type (TP).

(a) (6 points) Translate the following query into relational algebra: "What were the departure and arrival cities of all flights on December 10th that were completely full?" For the purposes of this one question, don't worry about efficiency.

$\pi_{DC,AC}(\sigma_{DT=12\text{-}10\text{-}01 \wedge NP=CA}(\textit{Flight} \bowtie (\textit{Route} \bowtie \textit{Plane}))$
or
$\pi_{DC,AC}(\sigma_{DT=12\text{-}10\text{-}01 \wedge NP=CA}((\textit{Flight} \bowtie \textit{Route}) \bowtie \textit{Plane})$.

NB: Plane $\bowtie$ *Flight does not work as a natural join: the relations have no field in common. Also, Plane* $\bowtie_{CA=NP}$ *Flight does not have the desired effect: it will pair tuples representing little planes with tuples representing partially full flights on big planes.*

(b) (5 points) Explain why the following cannot be expressed in relational algebra: "How many flights into Boston were late on December 10th?"

Relational algebra provides no way to compute new information; it only retrieves information already explicitly present in the database. The answer to this "query"

is not explicitly stored; it would have to be computed by counting the number of tuples returned by a more conventional query. One could easily imagine implementing a database capable of performing the counting operation, but it would step outside the realm of relational algebra to do so.

(c) (EXTRA CREDIT; 10 points) Assume that the Route and Flight relations have indices for attribute FN (only), the Plane relation has an index for attribute PM (only), and we don't want to bother creating any additional indices. Modify your answer to part (a) above by pushing selection and projection operations inward wherever it is both possible and profitable to do so.

$\pi_{DC,AC}(\sigma_{NP=CA}(Plane \bowtie \pi_{PM,NP,DC,AC}(\pi_{FN,NP}(\sigma_{DT=12\text{-}10\text{-}01}(Flight)) \bowtie Route)))$
or
$\pi_{DC,AC}(\sigma_{NP=CA}(\pi_{FN,NP}(\sigma_{DT=12\text{-}10\text{-}01}(Flight))$
$\bowtie \pi_{FN,CA,DC,AC}(Route \bowtie \pi_{PM,CA}Plane))).$

In both alternatives we can (and should) push the selection for day all the way in to the Flight relation. Because this forces us to iterate over the whole relation, the inner index join in the first suggested answer performs lookups on the Route relation. We therefore avoid pushing a project all the way in to Route. Because we are iterating over all elements of the result of the inner join (in order to produce them!) we do lookups on Plane in the outer index join, and avoid pushing a project in to Plane.

In the second suggested answer, the Plane relation is probably smaller than the Route relation, so we iterate over Plane and do lookups in Route. We therefore push a project in to Plane, but not in to Route. Because we have pushed a select in to Flight, we are forced to iterate over all tuples of both arguments of the outer join. The first suggested answer is probably the faster of the two.

6. (EXTRA CREDIT; 10 points) Prove that regular languages are closed under complementation. That is, given a finite automaton M that accepts certain strings composed of symbols from the input alphabet Γ, show that there exists a finite automaton M' that accepts all and only those strings over the same alphabet that are *not* accepted by M.

Assume without loss of generality that M is a deterministic machine with a dead state, so it always consumes its entire input (never gets "stuck"). Let M' be identical to M, except that all the accepting states in M' are non-accepting states in M, and all the non-accepting states in M' are accepting states in M. Clearly after consuming a given input string, M and M' are in the same state. But now M' accepts only if M does not, and vice versa.

NB: Replacing each edge label a with $\Gamma - \{a\}$ doesn't work. To see why not, consider a state with several outgoing transitions. Complementing the labels on each transition will produce overlapping label sets, introducing new paths, changing a DFA into an NFA, and changing the accepted language in complex and undesired ways.

7. (6 points) Determine, using a truth table, whether or not the following formula is a tautology: $(A \wedge (\neg B \vee C)) \vee (\neg C \wedge B) \vee \neg A$.

A	B	C	X $(\neg B \vee C)$	Y $(A \wedge X)$	Z $(\neg C \wedge B)$	$Y \vee Z \vee \neg A$
0	*0*	*0*	*1*	*0*	*0*	*1*
0	*0*	*1*	*1*	*0*	*0*	*1*
0	*1*	*0*	*0*	*0*	*1*	*1*
0	*1*	*1*	*1*	*0*	*0*	*1*
1	*0*	*0*	*1*	*1*	*0*	*1*
1	*0*	*1*	*1*	*1*	*0*	*1*
1	*1*	*0*	*0*	*0*	*1*	*1*
1	*1*	*1*	*1*	*1*	*0*	*1*

The formula is a tautology, because the right-hand column is all ones. BTW, this problem is straight off the 1999 final.

8. Consider the following quotation: "If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned."

   (a) (8 points) Restate the quotation in propositional logic, **as a collection of CNF terms**.

   Let A = mythical, B = mortal, C = mammal, and D = horned. Then

implications	*CNF*	*#*
$A \rightarrow \neg B$	$(\neg A \vee \neg B)$	*1*
$\neg A \rightarrow (B \wedge C)$	$(A \vee B)$	*2*
	$(A \vee C)$	*3*
$(\neg B \vee C) \rightarrow D$	$(B \vee D)$	*4*
	$(\neg C \vee D)$	*5*

   (b) (8 points) Prove, using resolution, that the unicorn is horned, or demonstrate that this cannot be proven from the given premises.

	justification	*#*
$(A \vee D)$	*resolution of 3 and 5*	*6*
$(\neg B \vee D)$	*resolution of 1 and 6*	*7*
(D)	*resolution of 4 and 7*	*8*

   (c) (8 points) Prove, using resolution, that the unicorn is mythical, or demonstrate that this cannot be proven from the given premises.

   By considering all pairs of clauses, we can exhaustively enumerate all CNF clauses that can be derived from our premises via resolution:

	justification	*#*
$(\neg B \vee C)$	*resolution of 1 and 3*	*9*
$(\neg A \vee D)$	*resolution of 1 and 4*	*10*
$(C \vee D)$	*resolution of 3 and 10*	*11*

   These clauses complete the list (except for (T), which we can derive from, say, 1 and 2). Since resolution is complete for propositional logic, and (A) does not appear in our list of clauses, we cannot prove that the unicorn is mythical.

Alternatively, we can show the unprovability of mythical *by giving a truth assignment in which* mythical *is false but all of the premises are true. Experimentation with a truth table reveals that there is exactly one such assignment: let* mythical *be false and* mortal, mammal, *and* horned *be true.*

NB: the fact that mythical *is not the right-hand side of any explicitly stated implication does* not *mean that it cannot be proven. If, for example, we know that* $\neg A \rightarrow (B \wedge \neg B)$*, then we can deduce* A*.*

9. (8 points) Characterize each of the following problems as (1) tractable (a polynomial-time algorithm exists), (2) intractable (best known algorithm requires exponential time), or (3) uncomputable.

   (a) Given a formula in propositional logic, does there exist a truth assignment for the propositions (a choice of which propositions are true and which are false) that makes the formula as a whole true?

   (b) Given a directed graph, a distinguished node v, and a constant k, list all nodes within distance k of v.

   (c) Given a program P, tell whether P runs in time $O(N)$, where N is the length of its input.

   (d) Given a finite automaton M and an input i, determine whether M accepts i.

   (b) and (d) are tractable. (a) is intractable. (c) is uncomputable.

10. (8 points) Translate the following English statements into predicate logic.

    (a) Not every day is cloudy.
    $(\exists d)[\neg \mathit{cloudy}(d)]$

    (b) Everyone has a cluster or a major in some department in the humanities.
    $(\forall p)(\exists d)[\mathit{humanities}(d) \wedge (\mathit{major}(p,d) \vee \mathit{cluster}(p,d))]$

    (c) There is something we all agree on.
    $(\exists t)(\forall p)[\mathit{agrees}(p,t)]$

    (d) Anyone who outperforms everyone else in the class will get an A.
    $(\forall a)[((\forall b)[(a = b) \vee \mathit{outperforms}(a,b)]) \rightarrow \mathit{gets\text{-}an\text{-}A}(a)]$

    NB: In several of these problems there is some ambiguity as to what should be expressed explicitly with a quantified variable, rather than being rolled into a predicate. I gave full credit only when all entities mentioned explicitly in the question were mentioned explicitly in the answer. In part (b), for example, I did not give full credit for $(\forall p)[\mathit{has\_humanities\_cluster}(p) \vee \mathit{has\_humanities\_major}(p)]$*. Note that in the limit, one could say* $(\forall p)[\mathit{has\_humanities\_cluster\_or\_major}(p)]$*, or even (rolling* p *itself into a zero-argument predicate) everybody_has_a_humanities_cluster_or_major, for which hopefully no one would expect full credit.*

    There's also some ambiguity as to what aspects of variables should be expressed as predicates and what can simply be considered a characteristic of the domain. In

part (a), for example, one could say $(\exists d)[day(d) \wedge \neg cloudy(d)]$*, but since we don't talk about things other than days, it's reasonable simply to say that the domain of* d *is the set of all days. In contrast, in (b) the wording implies the existence of departments that are* not *in the humanities, so it seems necessary to capture "humanities-ness" with a predicate.*