UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering and Computer Sciences
Computer Science Division

CS 164
Spring 1991
K. A. Yelick

CS 164: Programming Languages and Compilers
Midterm Exam

Name:

Problem 1 (25 pts total) In Mustang comments are currently specified by a percent sign, %. Some members of the Mustang user community have decided that the comment facility should be replaced to allow for a more general style. A comment now begins with /* and may contain any characters in any order, except that the */ sequence will close them. There is an exception to the closing rule: */ will not close the comment if it is bracketed by a pair of $'s. For example, in the following code the */ in the last line closes the comment.

```
/*
The old code is:
a := 9;
b := a;
c := 7;
print("About to do the computation");
$ newline;
doitnow(a,b) /* Do the program */ $
*/
```

This allows easier commenting-out of code which contains comments. The $ need not be immediately next to the comment, and they are paired like quotes.

Part A (5 pts) Write a context-free grammar for comment. You may use the grammar symbol char to represent a single character that is neither $, *, or /.

Part B (10 pts) A context free grammar is unnecessarily powerful. Write a regular expression for comments. (You may again use char, but do not make up other symbols.)

Part C (10 pts) Using Part B, one could augment the scanner to accept comments, or using Part A one could accepts comments through the parser. Which would you do and why?

Problem 2 (30 pts total) Graham R. Hacker (Alyssa's older brother, for those of you who took 60A) spent a few years working for a team of secret code crackers. One of his assignments was to reverse engineer an LALR parser to find an accepting string. He was given the following LALR parsing table, and partially readable list of production rules.

```
Action Goto
state ( ) d $ C S
0 S1 - S2 - 3 8
1 S4 - - - - -
2 R3 R3 R3 R3 - -
3 S1 - S2 - 5 -
4 S1 - S2 - 6 -
5 R1 R1 R1 R1 - -
6 - S7 - - - -
7 R2 R2 R2 R2 - -
8 - - - acc - -
```

Part A (10 pts) Draw the state transition diagram that is represented by this table. Label the transitions with the appropriate grammar symbol, and the states by their numbers. (You need not write the sets of items in each state.)

Part B (10 pts) There are three production rules in the grammar: the left-hand sides are readable, but the right-hand sides have been smudged so that only the number of symbols is visible. Fill in the right hand sides, writing one symbol in each place.

```
S --> _ _
C --> _ _ _ _
C --> _
```

Part C (2 pts) Give one example of a 5 character sentence in this language.

Part D (4 pts) How many shifts would be performed in parsing your sentence from part C?

Part E (4 pts) How many reduces would be performed in parsing your sentence from part C?

Problem 3 (20 pts total) Reah Covery just finished implementing her CS164 Mustang compiler. She implemented both phrase level and panic mode error recovery for parser, in addition to the error detection during scanning and static semantics checking.

Part A (10 pts) Her compiler produced the following error messages. For each one, state whether the error was most likely detected during scanning, phrase-level recovery, panic mode recovery, or static semantics checking. Justify your answer.

```
"line 5, column 12: ignoring extraneous semicolon"

"line 7, column 15: improperly formed statement"

"line 10, column 5: expected an array, found an int 'x'"

"line 12, column 20: illegal character '-['"
```

Part B (10 pts) In Reah's a phrase-level recovery, a trial parse ("parse-check") is employed to evaluate candidates for the error repair. Typically, the parser attempts to consume a bounded number of symbols, say K, beyond the point of error detection, after editing the token stream to reflect the repair. She knows what the value of K can affect the quality of recovery, either by being too small or too large. What considerations should she take into account in choosing K?

Problem 4 (25 pts total) Graham R. Hacker has given up espionage and is now working for a local software company, designing a static semantics checker for the block-structured language given below. Statements are represented by the nonterminal S, declarations are D, and blocks are B. A statement is made up of a operator and an identifier, and the typing rule is that in top may only be applied to an int id and realop may only be applied to a real id. (It isn't important what the operators do.) The usual scoping rules of block structured languages apply.

```
P --> B
B --> [ D ; G ]
D --> D id : int | D id : real | epsilon
G --> S G | B G | epsilon
```

```
S --> intop id | realop id
```

Part A (5 pts) Someone suggested the following program for testing Graham's static checker. Show that it is syntactically correct by drawing its parse tree.

```
[a: int b: real;
[a: real; intop a intop b]
[b: int b: int; realop a intop c]]
```

Part B (5 pts) Identify the static semantics errors by giving the kind of error, the line number (1-3), and the name of any identifier involved.

The chief engineer told Graham to give semantic rules that define the following three attributes.

```
D.DUPS is the set of identifiers that occur more than once
in D
S.UNDECL is the set of identifiers in S that have not been
declared
S.TYPERR is the set of identifiers in S that are used with
operators of the wrong type.
```

Graham had decided he needs another attribute to store declarations, and has chosen a global attribute (named T) that represents a block-structured symbol table like the one he implemented in CS164. Being a good software engineer, he wrote down a specification of the symbol table operations; it is included at the end of this problem. Graham has written the following semantic rules for a few of the productions.

```
P --> B {}

D1 --> D2 id : int {if isin_local(T, id.VAL)
then D1.DUPS := D2.DUPS U {id.VAL}
else D1.DUPS := D2.DUPS
insert(T, id.VAL, int)}

G1 --> S G2 {}

G1 --> B G2 {}

S --> intop id {if lookup(T, id.VAL) == NULLTYPE
then S.UNDECL := {id.VAL}
else if lookup(T,id.VAL) == real
then S.TYPERR := {id.VAL}}
```

Part C (10 pts) Write the semantic rules for the remaining productions. In each case the rules must appear at the right end of the production. (You may assume the symbol table is created before parsing begins.)

```
B --> [D ; G]

D1 --> D2 id : real

D --> epsilon

G --> epsilon

S --> realop id
```

Part D (5 pts) Using the program from Part A, show the sequence of operations that are performed on the symbol table T during evaluation.

Symbol table specification for reference.

A symtab is a mutable data type with operations: create, enterblock, exitblock, insert, lookup, and isin_local.

```
symtab create();
/* ensures: returns a new empty symbol table */

insert(symtab s, ident i, typename t);
/* ensures: adds the bindings [i, t] to s in the current
scope */

typename lookup(symtab s, ident i);
/* ensures: if i is not bound in s, then a special NULLTYPE
is
returned, otherwise, the type of the most recent binding
for i is
returned */

typename isin_local(symtab s, ident i);
/* ensures: returns true if i is declared in current scope,
false
otherwise */
```

```
enterblock(symtab s);
/* ensures: creates a new scope in s */

exitblock(symtab s);
/* ensures: removes the current scope from s *
```