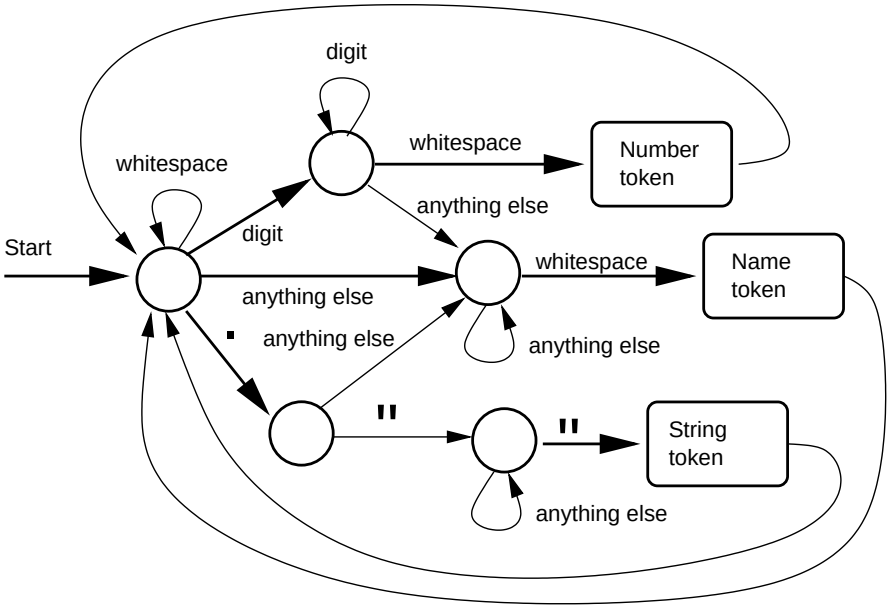
University of Toronto

CSC488S/CSC2107S Compilers & Interpreters **Winter 2003/2004**
Midterm Test Sample Solution **March 4, 2004**

1. [20 marks] Most solutions specified the lexical analyzer as a finite state machine. One possible machine is shown below.



Important points:

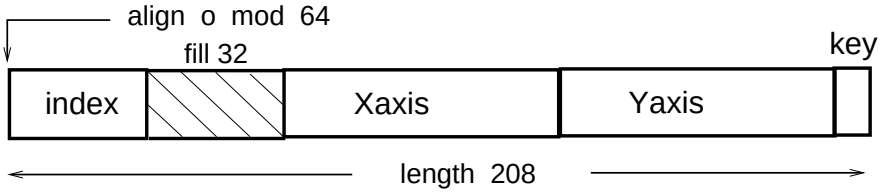
The digit part of the machine had to allow for non-digits and transfer to the name part of the machine.
`1234+9` is a valid name.

The string start `."` had to be handled as two characters. `.X` is a valid name.

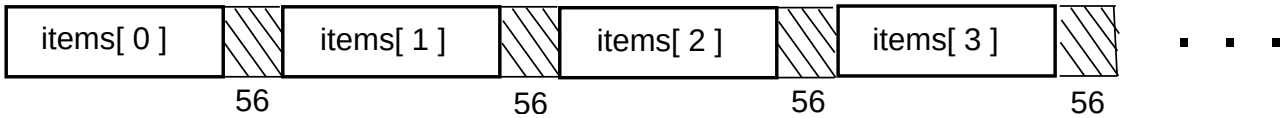
Names could contain *any* characters, not just letters. `A*B+c/7-(X/Y)])<` is a valid name.

The FSM had to be complete. Many people left off the return to the start state.

2. [20 marks] First layout the structure using the algorithm given in the lecture slides.



To make every element of the array *items* properly aligned, 56 bits of fill had to be added between the array elements.



Note that this makes the size of the array larger than 10 times the size of an array element.

3. [20 marks] The decoupling raises several issues.

- It really requires a two pass semantic analysis. One pass to collect all the declarations, a second pass to process use of the declarations. This also handles the case where P might call itself recursively (e.g. on line 16) before all of the parameter declarations have been seen.
- The parameter declarations in the procedure header can be treated like *forward* declarations. Enter the name into the procedures symbol table and mark it as an incomplete declaration.
- When a declaration for a parameter is encountered in the procedure body, the rest of the symbol table entry for the parameter is filled in.
- There must be a check that each parameter name declared in the procedure header is declared exactly once in the procedure body.

Someone asked if the language allowed declarations inside control constructs, e.g.

```
if( expn )
     double X ;
```

It does, but since the if statement doesn't automatically create a minor scope, the declaration is treated as if it was outside the if.

4. [20 marks] The grammars:

1. Not LL(1) since S has common start for 2 rules.
   Not LALR(1) due to reduce/reduce conflict for S
2. Not LL(1) since S has common start for 2 rules.
   LALR(1) since lookahead sets are disjoint (or by elimination).
3. ERROR, should have been LL(1) but C in the rule for B is wrong.
   Everyone received 5 marks for 4.3
4. Not LL(1) since B has common start for 2 rules.
   Not LALR(1) due to shift/reduce conflict for B

5.[20 marks] A correct solution had to describe

a) How overloaded methods were stored in the symbol table. The simplest solution is to create an entry for each member function and link together member functions with the same name.

b) How to add new member functions to the symbol table. There had to be a semantic analysis check that a function with exactly the same signature wasn't already in the table.

c) There had to be an algorithm for finding the correct member function for a given call. The simplest solution is to work through the linked list of member functions with the same name, looking for a matching parameter signature. Note that the signature involved the *type* of each parameter, not just the number of parameters. Two functions with the same number of parameters, but different type, e.g. `P( int A )` and `P( bool B )` is legal.

   A few people pointed out that the type of an actual parameter might be ambiguous due to automatic type conversion rules.
   For example `P( 1.0 )` could match `P( float X )` or `P( double Y )` .