CSC 306 WEEK 9 TEST SOLUTIONS

PROFESSOR GODFREY MUGANDA

1. Fill in the blanks and Short Answer (18%)

1. Suppose that a C++ class is called `MyClass`:

   (a) What is the type of the parameters of the default constructor for `MyClass`?
   Default Constructors have no parameters.
   (b) What is the type of the parameters of a convert constructor for `MyClass`?
   Any type other than the class type.
   (c) What is the type of the parameters for the copy constructor for `MyClass`?
   Reference to the same class type: `MyClass &`

2. When a C++ programmer assigns the value returned by the `new` operator to a pointer and then forgets to perform a `delete` on the pointer after he or she is done using the allocated memory, the program will suffer from a problem known as *a memory leak.*

3. Suppose a C++ programmer has two pointers `p` and `q` pointing to the same memory that has been allocated by a call to `new` and then the programmer performs a `delete` on the pointer `p`. If the programmer then attempts to use the deleted memory through `q`, we have a problem known as the *dangling pointer* problem.

4. In C++, A programmer tells the compiler that a member function of a class should have polymorphic behavior by tagging the member function with the keyword *virtual.*

5. While building the internal representation of a program during the development of an interpreter or compiler, the variable that tracks the position in the statement table where the next statement will be loaded is called *location counter.*

6. During the execution phase of a program by an interpreter, the variable that tracks the position in the statement table of the next statement to be executed is called *program counter.*

7. Unlike Java, C++ does not have an *abstract* keyword to denote the fact that a a class method has no implementation. What is the equivalent notation in C++ that denotes an abstract member function?

Use a prefix of `virtual` *on the declaration of the member function and add a suffix of* `= 0`*, as in*

```
virtual void mymethod() = 0;
```

2. More Short Answer (16%)

8. Give an example of the use of *dynamic method binding* in our implementation of the NINC interpreter, and explain why it was necessary to use dynamic binding in that example.

One example is the EXPR class which has many different subclasses. Each subclass has its own `tostring()` *member function that returns a string representation of an object of that class. All these* `tostring()` *methods are called through a pointer to EXPR. Dynamic binding makes it possible to call the member function in the type of the subclass object rather than the type of the pointer.*

9. (a) One of the casts available in C++ is `dynamic_cast`. Explain under what conditions a dynamic cast should be used. Be sure to include how the value returned from a dynamic cast is used by the programmer.

A dynamic cast is used in the context of an inheritance hierarchy when a super class pointer is pointing to a subclass object and you do not know the exact class of the object being pointed to. You use a dynamic cast to cast the pointer to some type `X` *you think it should be: If the cast succeeds, you get the pointer to the subclass object of type* `X`*. If the cast fails, you get the value* `nullptr`*.*

9. (b) Give a short fragment of code that shows how to use a dynamic cast, and which illustrates your answer to part (a) above.

Suppose that `Dog` and `Cat` are subclasses of `Animal`, and that `Cat` has a `meow()` method while `Dog` has a `bark()` method.

If you have a pointer `pA` to `Animal` that points to something that could be dog or a cat, you can write

```
Cat *pC = dynamic_cast<Cat *>(pA);
if (pC != nullptr) pC -> meow();   // It is a cat
Dog *pD = dynamic_cast<Dog *>(pA);
if (pD != nullptr) pD -> bark();   // It is a dog
```

9. (c) How can a Java programmer mimic the functionality of C++'s `dynamic_cast` facility?

Use the `instanceof` *operator. For example, in the same context as above,*

```
if (pA instance of Dog)
{
   pD = (Dog)pA;
   pD.bark();
}
if (pA instance of Cat)
{
   pC = (Cat)pA;
   pC.meow();
}
```

3. NINC Internal Representation

10. (6 %) For the internal representation of a NINC program, we defined a class

```
class CondGotoSTMT
```

with two fields:

```
int target;
EXPR *pExpr;
```

Assuming you can access a global variable

```
int pc;
```

that represents the program counter, Write a class member function

```
void execute()
```

that uses these two fields to execute the *conditional goto* statement at runtime.

The conditional go to statement exists to implement the while statement, where control transfers to the target statement if the expression evaluates to false. It is executed like this:

```
void execute()
{
   if (pExpr->eval() == false)
   {
      pc = target;
   }
}
```

11. (10 %) For the internal representation of a NINC program, we defined a class

```
class SwitchSTMT
```

so that it has three fields:

```
int target;
EXPR *pExpr;
map<int, int> jumpTable;
```

Assuming you can access a global variable

```
int pc;
```

that represents the program counter, Write a class member function

```
void execute()
```

that uses these three fields to execute the *switch* statement at runtime.

Hint:

A C++ map stores (key-value) pairs. Given a key `key`, the code

```
map<int, int>::iterator it = jumpTable.find(key);
```

returns an iterator equal to `jumpTable.end()` if the key is not found in the map. If the key is found, the the expression `it->first` gives the key part of the key-value pair(which is just the key) while the expression `it->second` gives the value part of the key-value pair.

Alternatively, if you know that the key is in the map, `jumpTable[key]` will give the associated value. Do not use the expression `jumpTable[key]` unless you know the key is in the jump table.

A switch statement is executed by using the value of the expression as a key into the jump table. If the key is found in the jump table, the program counter is set to the value associated with the key:

```
pc = jumptable[key];
```

but if the key is not in the jump table, then the program counter is set to target field of the switch statement:

```
pc = target;
```

To find out if the value of the expression is one of the keys in the jump table, use the `find` *member function of the* `map` *class.*

```
int key = pExpr->eval();
map<int, int>::iterator it = jumptable.find(key);
if (it == jumptable.end())
{
   pc = target;
}
else
{
   pc = jumptable[key];
}
```

4. Parsing

12. (10 %) Consider the grammar

$$\begin{array}{lcl} S & \rightarrow & (L) \quad | \quad a \\ L & \rightarrow & L, S \quad | \quad S \end{array}$$

where S is the start symbol of the grammar. Draw the parse trees for

(i) (a, a)

Hard to draw the parse tree, but you can get from the following derivation:

$$S \Rightarrow (L) \Rightarrow (L, S) \Rightarrow (S, S) \Rightarrow (a, S) \Rightarrow (a, a)$$

(ii) $(a, (a, a))$

Hard to draw the parse tree, but you can get from the following derivation:

$$\begin{array}{lllllll} S & \Rightarrow & (L) & \Rightarrow & (L, S) & \Rightarrow & (S, S) \quad \Rightarrow \quad (a, S) \\ & \Rightarrow & (a, (L)) & \Rightarrow & (a, (L, S)) & \Rightarrow & (a, (S, S)) \\ & \Rightarrow & (a, (a, S)) & \Rightarrow & (a, (a, a)) & & \end{array}$$

13. (40%) Consider the grammar

PROG ::= { DECS } #
DECS ::= VARDEC | FUNDEC
VARDEC ::= var d {, d } : TYPE;
TYPE ::= int | bool
FUNDEC ::= function id ( ) begin s {; s } end

Assume that the terminal symbols in this grammar are defined by an enumeration type

```
enum class Token
{
   FUNCTION, ID, VAR, INT, BOOL, BEGIN, END,
   D, S, COMMA,
   COLON, SEMICOLON, LPAREN, RPAREN, EOS
};
```

where EOS represents the end of stream marker #.

Assume also that there are functions

```
Token getToken()
void accept(Token t)
```

that will return the next token from the source file; and verify the presence of an expected token and skip that token. Assume further that there is a global variable

```
Token token
```

that can be used as the lookahead token during the parsing process. Write a recursive descent parser for the grammar.

```
void prog()
{
   while (token != Token::EOS)
   {
      decs();
   }
}


void decs()
{
  switch(token)
  {
     case Token::VAR: vardec(); return;
     case Token::FUNCTION: fundec(); return;
     default: error();
  }
}




void vardec()
{
   accept(Token::VAR); accept(Token::D);
   while (token == Token::COMMA)
   {
     token = getToken();
     accept(Token::D)
   }
```

```
   accept(Token::COMMA);
   type();
   accept(Token::SEMICOLON);
}


void type()
{
   if (token == Token::INT || token == Token::BOOL)
     token = getToken();
   else
     error();
}


void fundec()
{
   accept(Token::FUNCTION);  accept(Token::ID);
   accept(Token::LPAREN); accept(Token::RPAREN);
   accept(Token::BEGIN); accept(Token::S);
   while (token == Token::SEMICOLON)
   {
     token = getToken();
     accept(Token::S);
   }
   accept(Token::END);
}
```