1. (20 points) For the following program, write functions found and display. Function found, used on line 11, returns true if the string is in the map, and false otherwise. Function display, used on line 15, prints the key and value for each item in the map. A sample test is provided, lines 11–15, but your solution should not depend on the actual parameters to the function. Sample output:

```
found
(i, 77)   (sum, 83)   (value, 86)
```

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <string>
4  #include <map>
5
6  void display(const std::map<std::string, int> & myMap) {
7    for (const auto& pear : myMap) {
8      std::cout << '(' << pear.first << ",␣" << pear.second
9                << ')' << "␣␣␣";
10   }
11   std::cout << std::endl;
12 }
13
14 bool altFound(const std::map<std::string, int> & myMap, std::string s) {
15   return myMap.count(s) != 0;
16 }
17
18 bool found(const std::map<std::string, int> & myMap, std::string s) {
19   std::map<std::string, int>::const_iterator it = myMap.find(s);
20   if ( it == myMap.end() ) return true;
21   else return false;
22 }
23
24 int main() {
25   std::map<std::string, int> myMap;
26   myMap["sum"] = rand()%100;
27   myMap["value"] = rand()%100;
28   myMap["i"] = rand()%100;
29
30   if ( altFound(myMap, "snorlax") || altFound(myMap, "value") ) {
31     std::cout << "found" << std::endl;
32   }
33   else  std::cout << "Not␣there" << std::endl;
34   display( myMap );
35 }
```

2. (10 points) There are four files listed below: two for class A and two for class B. However, the two classes mutually depend on each other. Add code so that main.cpp will compile.

****************************** a.h ******************************

```
#include <iostream>
#include "b.h"

class A {
public:
  A() : b() { }
  void f();
  int g();
private:
  B b;
};
```

****************************** a.cpp ******************************

```
#include "a.h"
void A::f() { b.makeA(); b.g(); }
int A::g() { return 99; }
```

****************************** b.h ******************************

```
class A;

class B {
public:
  B();
  int g() const;
  void makeA();
private:
  A* a;
};
```

****************************** b.cpp ******************************

```
#include <iostream>
#include "a.h"

B::B() { }
int B::g() const { std::cout << "B::g()" << std::endl; }
void B::makeA() { a = new A; }
```

**************************** main.cpp ****************************

```
#include <iostream>
#include "a.h"

int main() {
  A a;
```

```
    a.f();
}
```

3. (10 points) For the parser in the following program, non-terminals are represented by lower case letters, and terminals are represented by upper case letters. Give the output; assume the user types: a.

```
%{
#include "parse.tab.h"
%}
%option noyywrap
%%
"a"    { return A;}
"\n"   { }
.                    { }
```

```
%{
#include <iostream>
extern int yylex();
void yyerror(const char * msg) { std::cout << msg << std::endl; }
%}
%token A

%%
start    : x { std::cout << "x" << std::endl; }
x        : y { std::cout << "y" << std::endl; }
y        : z { std::cout << "z" << std::endl; }
z        : A { std::cout << "A" << std::endl; }
```

```
#include <iostream>
extern int yyparse();
int main() {
  if ( yyparse() == 0 ) { std::cout << "Syntactically correct." << std::endl; }
  else { std::cout << "Oops!" << std::endl; }
}
```

```
a
A
z
y
x
```

---

4. (10 points) Listed below is a scanner and parser for matching balanced parentheses. However, semantic actions have been inserted in the middle of the rule on line 16. Give the output for the program, assuming that the user types (()())() and then hits **[enter]**.

```
 1  %{
 2  #include "parse.tab.h"
 3  %}
 4
 5  %%
 6
 7  "("        { return LPAR; }
 8  ")"        { return RPAR; }
 9  "\n"       { return CR; }
10  .          { ; }
11
12  %%
13  int yywrap() { return 1; }
```

```
 1  %{
 2  #include <iostream>
 3  extern int yylex();
 4  void yyerror(const char * msg) { std::cout << msg << std::endl; }
 5  %}
 6  %token CR LPAR RPAR
 7  %%
 8  lines    : lines expr CR
 9             {
10               std::cout << std::endl;
11               std::cout << "accept" << std::endl;
12             }
13           | %empty
14           ;
15  expr     : expr expr
16           | LPAR {std::cout << "(";} expr RPAR
17                  {std::cout << ")";}
18           | LPAR RPAR
19             { std::cout << "(*)" ; }
20           ;
```

```
((*)(*))(*)
accept
```

---

5. (10 points) A scanner and parser are listed below. Insert semantic actions into the scanner and parser so that after integers are inserted, followed by newline, the sum of the numbers is printed. You may not insert any global variables. Sample output might be:

```
34 22 11 99
sum is 166
```

```
%{
#include "parse.tab.h"
%}
%%

[0-9]+      { yylval = atoi(yytext);   return NUMBER; }
"\n"        { return CR; }
.           { ; }

%%
int yywrap() { return 1; }
```

```
%{
#include <iostream>
extern int yylex();
void yyerror(const char * msg) { std::cout << msg << std::endl; }
%}
%token NUMBER CR
%%

lines     : lines numbers CR
            { std::cout << "sum is " << $2 << std::endl; }
          | %empty
          ;

numbers    : numbers number
            { $$ = $1 + $2; }
          | %empty
            { $$ = 0; }
          ;

number     : NUMBER
            { $$ = $1; }
```

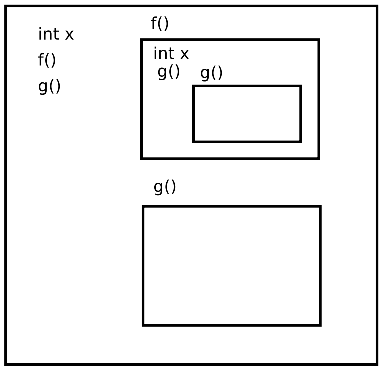
6. (20 points) Below is a Python program. Give the output for the program and write a contour diagram illustrating the scopes of the program. Then draw the chain of symbol tables needed to implement the scope rules at the deepest nesting level. Explain why the program works. In the event that you think the program doesn't work, explain why it doesn't work.

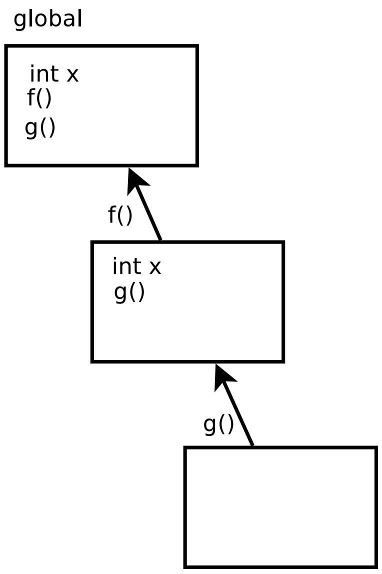
```
def f():
   def g():
      print x
   x = 4
   g()

def g():
   print x

x = 99
x += 1
f()
print x
g()
```

```
4
100
100
```





7. (20 points) Listed below is a scanner and parser intended to parse a sequence of space-separated numbers. Modify the scanner and insert semantic values & actions so that for every line of numbers input, the numbers are then printed to standard output in sorted order. You may not insert global variables. However, you may assume a function, ‘magicsort’, is provided with the signature given on line 8 of the parser. ‘magicsort’ can be passed a vector by reference, after which the vector can be assumed sorted. Sample input and output:

```
44 23 99 10 3
3 10 23 44 99
```

```
%{
#include <iostream>
#include <vector>
#include <cstring>
#include "parse.tab.h"
%}

digits      [0-9]+

%%

{digits}   { yylval.val = atoi(yytext);  return NUMBER; }
"\n"       { return CR; }
.          { ; }

%%
int yywrap() {
   yylex_destroy();
   return 1;
}
```

```
%{
  #include <iostream>
  #include <algorithm>
  #include <vector>

extern int yylex();
void yyerror(const char * msg) { std::cout << msg << std::endl; }
std::string output = std::string();
void printSortedNumbers( std::vector<int>* );
%}

%union {
  int val;
  std::vector<int>* vec;
}

%token NUMBER CR

%type<vec> numbers lines
%type<val> number NUMBER

%%

lines    : numbers CR {
               printSortedNumbers($1);
               delete $1;
             }
           ;

numbers : numbers number
             { $$->push_back($2); }
           | %empty
             { $$ = new std::vector<int>(); }
           ;

number   : NUMBER
             { $$ = $1; }

%%

void printSortedNumbers(std::vector<int>* vec) {
  std::sort( (*vec).begin(), (*vec).end() );
  for (unsigned int i = 0; i < (*vec).size(); ++i) {
    std::cout << (*vec)[i] << std::endl;
  }
}
```