1. (20 points) Give the output for the following Python 2.7 program.

```
1  print -1/2
2  print -1/2 -1/2
3  print -1/2 + -1/2
4  print -5%2
5  print 5%-2
```

---

2. (20 points) Give short answers for the following:

   (a) What is the goal of parsing.

   (b) Can the following grammar be used to implement a top-down parser? Why or why not?

```
E   : E '+' T | T
T   : T '*' F | F
F   : ( E ) | ID
```

   (c) Given a grammar in LL(1) format, how would you write a *recursive descent* parser. Since a grammar consists of terminals and non-terminals, simply explain how you would handle each in writing your parser.

   (d) Bison is an LALR(1) parser. What does the 1 indicate?

3. (20 points) Give the output of the program listed below and on subsequent pages.

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

```
1  #include <iostream>
2  #include <list>
3  #include "shape.h"
4  #include "visitor.h"
5
6  float getArea(const std::list<Shape*> & shapes,
7                AverageAreaVisitor & areaVisitor) {
8    for ( Shape* shape : shapes ) {
9      shape->accept(&areaVisitor);
10   }
11   return areaVisitor.getAvg();
12 }
13
14 int main() {
15   std::list<Shape *> shapes;
16   shapes.push_back(new Circle(3, 3, 5));
17   shapes.push_back(new Rectangle(0, 0, 3, 4));
18   AverageAreaVisitor areaVisitor;
19   float area = getArea(shapes, areaVisitor);
20   std::cout << "area of all shapes is: " << area << std::endl;
21 }
```

********** shape.h **********

```
#include "vector2f.h"

class Shape {
public:
  Shape(float x, float y) : position(x, y) {}
  virtual void accept(class Visitor * const) = 0;
private:
  Vector2f position;
};

class Rectangle : public Shape {
public:
  Rectangle(float x, float y, float w, float h) :
    Shape(x, y), width(w), height(h) {}
  float getWidth() const { return width; }
  float getHeight() const { return height; }
  virtual void accept(Visitor * const v);
private:
  float width, height;
};

class Circle : public Shape {
public:
  Circle(float x, float y, float r) : Shape(x, y), radius(r) {}
  float getRadius() const { return radius; }
  virtual void accept(Visitor * const v) ;
private:
  float radius;
};
```

********** shape.cpp **********

```
#include <iostream>
#include "shape.h"
#include "visitor.h"

void Rectangle::accept(Visitor * const v) {
  std::cout << "In Rectangle::accept" << std::endl;
  v->visit(this);
}
void Circle::accept(Visitor * const v) {
  std::cout << "In Cirle::accept" << std::endl;
  v->visit(this);
}
```

********** visitor.h **********

```
class Visitor {
public:
  virtual void visit(const class Rectangle * ) = 0;
  virtual void visit(const class Circle * ) = 0;
};

class AverageAreaVisitor : public Visitor {
public:
  AverageAreaVisitor() : sum(0), count(0) {}
  virtual void visit(const Rectangle * ) ;
  virtual void visit(const Circle * ) ;
  float getAvg() const { return (count > 0) ? sum/count : 0; }
private:
  float sum;
  unsigned int count;
};
```

********** visitor.cpp **********

```
#include "cmath"
#include "visitor.h"
#include "shape.h"

void AverageAreaVisitor::visit(const Rectangle *) {
  std::cout << "In Rectangle::visit" << std::endl;
   sum += 25;
   ++count;
}

void AverageAreaVisitor::visit(const Circle *) {
  std::cout << "In Circle::visit" << std::endl;
   sum += 10;
   ++count;
}
```

4. (20 points) Convert class SymbolTable into a GoF singleton; also, convert function main to use the singleton. (hint: four steps for GoF: move constructor to private, declare a static instance variable, define the static variable, and write function getInstance())

```
#include <iostream>
#include <string>
#include <map>

class SymbolTable {
public:
  SymbolTable() {}
  void setValue(const std::string& name, int val) { table[name] = val; }
  int  getValue(const std::string& name) const;
private:
  std::map<std::string, int> table;
};
int SymbolTable::getValue(const std::string& name) const {
  std::map<std::string, int>::const_iterator it = table.find(name);
  if ( it == table.end() ) throw name+std::string(" not found");
  return it->second;
}
```

```
#include "symbolTable.h"

int main() {
  SymbolTable table;
  table.setValue("sum", 17);
  std::cout << table.getValue("sum") << std::endl;;
}
```

5. Using the symbol table from the previous problem, together with the scanner and ast specified on the following pages, insert semantic actions into the bison parser specification listed below so that the program builds an AST, performs addition and subtraction on variable and integer expressions, and prints the results of the evaluation.

```
%{
#include <iostream>
#include <map>
#include <cmath>
#include "ast.h"
extern int yylex();
void yyerror(const char * msg);
%}

%union {
  Node* node;
  int number;
  char *id;
}
%token CR EQ
%token<id> IDENT
%token<number> INT
%type<node> lines expr
%left PLUS MINUS
%left MULT DIV

%%
lines   : lines expr CR


        | lines IDENT EQ expr CR


        | lines CR


        | { ; }
        ;

expr    : expr PLUS expr


        | expr MINUS expr


        | INT


        | IDENT


        ;
%%
void yyerror(const char * msg) { std::cout << msg << std::endl; }
```

```
************************* scanner *******************************
%{
#include "ast.h"
#include "parse.tab.h"
%}
letter      [a-zA-Z]
digit       [0-9]
ident       {letter}({letter}|{digit})*

%%

"="         { return EQ;    }
"+"         { return PLUS;  }
"-"         { return MINUS; }
"*"         { return MULT;  }
"/"         { return DIV;   }
{digit}+    {
              yylval.number = atoi(yytext);
              return INT;
            }
{ident}     {
              yylval.id = new char[yyleng+1];
              strcpy(yylval.id, yytext);
              return IDENT;
            }
"\n"        { return CR;    }
.           {}
%%
int yywrap() {
  yylex_destroy();
  return 1;
}
```

```
************************ ast.h *********************************
#pragma once
#include <string>
#include <map>
class SymbolTable;
class Node {
public:
  Node() {}
  virtual ~Node() {}
  virtual int eval() const = 0;
};

void freeAST(Node*);
class LiteralNode : public Node {
public:
  LiteralNode(int v) : Node(), val(v) { }
  virtual ~LiteralNode() {}
  virtual int eval() const { return val; }
private:
  int val;
};
class IdentNode : public Node {
public:
  IdentNode(const std::string id) : Node(), ident(id) { }
  virtual ~IdentNode() {}
  const std::string getIdent() const { return ident; }
  virtual int eval() const;
private:
  std::string ident;
};
class BinaryNode : public Node {
public:
  BinaryNode(Node* l, Node* r) : Node(), left(l), right(r) {}
  virtual int eval() const = 0;
  Node* getLeft()  const { return left; }
  Node* getRight() const { return right; }
protected:
  Node *left;
  Node *right;
};
class AssBinaryNode : public BinaryNode {
public:
  AssBinaryNode(Node* left, Node* right);
  virtual int eval() const;
};
class AddBinaryNode : public BinaryNode {
public:
  AddBinaryNode(Node* left, Node* right) : BinaryNode(left, right) { }
  virtual int eval() const;
};
class SubBinaryNode : public BinaryNode {
public:
  SubBinaryNode(Node* left, Node* right) : BinaryNode(left, right) { }
  virtual int eval() const;
};
```

```
************************ ast.cpp *********************************
#include <iostream>
#include <sstream>
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include "ast.h"
#include "symbolTable.h"

void freeAST(Node* node) {
  if ( node ) {
    BinaryNode* temp = dynamic_cast<BinaryNode*>(node);
    if ( temp ) {
      freeAST(temp->getLeft());
      freeAST(temp->getRight());
    }
    delete node;
  }
}
int IdentNode::eval() const {
  int val = SymbolTable::getInstance().getValue(ident);
  return val;
}

AssBinaryNode::AssBinaryNode(Node* left, Node* right) :
  BinaryNode(left, right) {
  int res = right->eval();
  const std::string n = static_cast<IdentNode*>(left)->getIdent();
  SymbolTable::getInstance().setValue(n, res);
}
int AssBinaryNode::eval() const {
  if (!left || !right) {
    throw "error";
  }
  int res = right->eval();

  const std::string n = static_cast<IdentNode*>(left)->getIdent();
  SymbolTable::getInstance().setValue(n, res);
  return res;
}
int AddBinaryNode::eval() const {
  if (!left || !right) {
    throw "error";
  }
  int x = left->eval();
  int y = right->eval();
  return (x+y);
}

int SubBinaryNode::eval() const {
  if (!left || !right) {
    throw "error";
  }
  int x = left->eval();
  int y = right->eval();
  return (x-y);
}
```