1. (10 points) Give the output for the following program.

```
#include <iostream>
#include <vector>

class Token {
public:
  Token()              { std::cout << "default" << std::endl;    }
  Token(const char*)   { std::cout << "convert" << std::endl;    }
  Token(const Token&){ std::cout << "copy" << std::endl;       }
  Token& operator=(const Token&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};

int main() {
  std::vector<Token> tokens;
  tokens.push_back( "IDENT" );
  tokens.push_back( "FLOAT" );
}
```

```
convert
copy
convert
copy
copy
```

---

2. (10 points) Give the output for the following program. Notice the use of *reserve* and *emplace_back*.

```
#include <iostream>
#include <vector>

class Token {
public:
  Token()              { std::cout << "default" << std::endl;    }
  Token(const char*)   { std::cout << "convert" << std::endl;    }
  Token(const Token&){ std::cout << "copy" << std::endl;       }
  Token& operator=(const Token&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
};

int main() {
  std::vector<Token> tokens;
  tokens.reserve(2);
  tokens.emplace_back( "IDENT" );
  tokens.emplace_back( "FLOAT" );
}
```

```
convert
convert
```

3. (10 points) Give the output for the following program.

```
1  #include <iostream>
2  #include <vector>
3
4  class Shape {
5  public:
6    void display() const {
7      std::cout << "SHAPE" << std::endl;
8    }
9    virtual void print() const {
10     std::cout << "BASE" << std::endl;
11   }
12   void doit(const Shape* shape) const {
13     shape->print();
14   }
15 };
16
17 class Circle : public Shape {
18 public:
19   void display() const {
20     std::cout << "CIRCLE: " << std::endl;
21   }
22   void print() const {
23     std::cout << "DERIVED: " << std::endl;
24   }
25 };
26
27 int main() {
28   Shape* shape = new Circle;
29   shape->display();
30   shape->print();
31   shape->doit(shape);
32 }
```

```
SHAPE
DERIVED:
DERIVED:
```

---

4. (15 points) *wc* is a unix utility that prints the number of lines, words, and bytes in an input stream or file. Insert actions into the scanner below so that main prints counts of lines, words, and bytes.

```
%{
#include <iostream>
  int lines = 0;
  int words = 0;
  int bytes = 0;
%}
letter      [a-zA-Z]
word        {letter}+

%%
{word}        { ++words; bytes += yyleng; }
"\n"          { ++lines; ++bytes;         }
.             { ++bytes;                  }
%%
int yywrap()  {
  yylex_destroy();
  return 1;
}
```

```
#include <iostream>
int yylex();

extern int words;
extern int lines;
extern int bytes;

int main() {
  yylex();
  std::cout << "words: " << words << std::endl;
  std::cout << "lines: " << lines << std::endl;
  std::cout << "bytes: " << bytes << std::endl;
  return 0;
}
```

5. (15 points) Insert actions into the parser so that for each line of input, the number of pairs of parentheses is printed. You may not define any global variables.

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

lines    : lines expr CR
           { std::cout << "parens: " << $2<< std::endl; }
         | { ; }
         ;

expr     : LPAR expr RPAR expr
           { $$ = $2 + $4 + 1; }
         | { $$ = 0; }
         ;
```

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

%%

"("        { return LPAR; }
")"        { return RPAR; }
"\n"       { return CR; }
.          { ; }

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

6. (15 points) Insert actions into the parser and scanner generators listed below so that the resulting program finds the sum of the numbers entered on a line of input. You may not define any global variables. For example, if the input is 23 45 22, the output should be Sum is: 90:

```
23 45 22
Sum is: 90
```

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

lines    : lines expr CR
           { std::cout << "Sum is: " << $2 << std::endl; }
         | { ; }
         ;

expr     : NUMBER expr
           { $$ = $1 + $2; }
         | { $$ = 0; }
         ;
```

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

number     [0-9]+

%%

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

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

7. (15 points) Insert actions into the parser and scanner generators listed below so that for each line of input, the arithmetic result is printed. You may not define any global variables.

```
%{
#include <iostream>
extern int yylex();
extern int yylval;
void yyerror(const char * msg);
%}
%token CR NUMBER PLUS MINUS MULT DIV
%%
lines    : lines expr CR
           { std::cout << "result: " << $2 << std::endl;  }
         | lines CR
         |
         ;
expr     : expr PLUS term
           { $$ = $1 + $3;  }
         | expr MINUS term
           { $$ = $1 - $3;  }
         | term
           { $$ = $1;       }
         ;
term     : term MULT factor
           { $$ = $1 * $3;  }
         | term DIV factor
           { $$ = $1 / $3;  }
         | factor
           { $$ = $1;  }
         ;
factor   : NUMBER
         ;
%%
void yyerror(const char * msg) { std::cout << msg << std::endl; }
```

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

%%

"+"        { return PLUS; }
"-"        { return MINUS; }
"*"        { return MULT; }
"/"        { return DIV; }
[0-9]+     { yylval = atoi(yytext);
             return NUMBER;
           }
"\n"       { return CR; }
<<EOF>>    { yyterminate(); }

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

8. (10 points) Insert code into the scanner listed below so that it uses the singleton, WordCount, to count the number of words in quotes. The main program also uses WordCount to print the count.

```
class WordCount {
public:
  static WordCount* getInstance() {
    if ( !instance ) instance = new WordCount;
    return instance;
  }
  void incrCount()      { ++count;  }
  int getCount() const { return count; }
private:
  static WordCount* instance;
  int count;
  WordCount() : count(0) {}
};
```

```
%{
#include "wordCount.h"
%}
word          [a-zA-Z]+
%x START

%%
\"              { BEGIN(START);          }
<START>{word}   { WordCount::getInstance()->incrCount(); }
<START>[^"]     {                        }
<START>["]      { BEGIN(INITIAL);        }

{word}          {                        }
.               {                        }
"\n"            {                        }
%%
int yywrap() { return 1; }
```

```
#include <iostream>
#include "wordCount.h"
int yylex();
WordCount* WordCount::instance = nullptr;
int main() {
  yylex();
  std::cout << "Number of words in quotes: "
            << WordCount::getInstance()->getCount()
            << std::endl;
}
```