

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

```
1 #include <iostream>
2 #include <vector>
3
4 class Token {
5 public:
6     Token() { std::cout << "default" << std::endl; }
7     Token(const char*) { std::cout << "convert" << std::endl; }
8     Token(const Token&){ std::cout << "copy" << std::endl; }
9     Token& operator=(const Token&) {
10         std::cout << "assign" << std::endl;
11         return *this;
12     }
13 };
14
15 int main() {
16     std::vector<Token> tokens;
17     tokens.push_back( "IDENT" );
18     tokens.push_back( "FLOAT" );
19 }
```

```
convert
copy
convert
copy
copy
```

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

```
1 #include <iostream>
2 #include <vector>
3
4 class Token {
5 public:
6     Token() { std::cout << "default" << std::endl; }
7     Token(const char*) { std::cout << "convert" << std::endl; }
8     Token(const Token&){ std::cout << "copy" << std::endl; }
9     Token& operator=(const Token&) {
10         std::cout << "assign" << std::endl;
11         return *this;
12     }
13 };
14
15 int main() {
16     std::vector<Token> tokens;
17     tokens.reserve(2);
18     tokens.emplace_back( "IDENT" );
19     tokens.emplace_back( "FLOAT" );
20 }
```

```
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.

```
1  %{
2  #include <iostream>
3      int lines = 0;
4      int words = 0;
5      int chars = 0;
6  %{
7  word      [a-zA-Z]+
8
9  %%
10 {word}      { ++words; chars += strlen(yytext); }
11 "\n"        { ++lines; ++chars; }
12 .           { chars += strlen(yytext); }
13 %%
14 int yywrap() { return 1; }
```

```
1  #include <iostream>
2  int yylex();
3
4  extern int words;
5  extern int lines;
6  extern int chars;
7
8  int main() {
9      yylex();
10     std::cout << "words: " << words << std::endl;
11     std::cout << "lines: " << lines << std::endl;
12     std::cout << "chars: " << chars << std::endl;
13     return 0;
14 }
```

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.

```
1  %{
2  #include <iostream>
3  extern int yylex();
4  void yyerror(const char * msg) { std::cout << msg << std::endl; }
5  %{
6  %token CR
7  %token LPAR
8  %token RPAR
9  %%
10
11 lines    : lines expr CR
12           { std::cout << "parens: " << $2 << std::endl; }
13           | { ; }
14           ;
15
16 expr     : LPAR expr RPAR expr
17           { $$ = $2 + $4 + 1; }
18           | { $$ = 0; }
19           ;
```

```
1  %{
2  #include <iostream>
3  #include <cstring>
4  #include "parse.tab.h"
5  %{
6
7  %%
8
9  "("      { return LPAR; }
10 ")"      { return RPAR; }
11 "\\n"    { return CR; }
12 "."      { ; }
13
14 %%
15 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
```

```
1  %{
2  #include <iostream>
3  extern int yylex();
4  void yyerror(const char * msg) { std::cout << msg << std::endl; }
5  %{
6  %token NUMBER CR
7  %%
8
9  lines      : lines expr CR
10             { std::cout << "Sum is: " << $2 << std::endl;
11               $$ = 0;
12             }
13             | { ; }
14             ;
15
16 expr       : NUMBER expr
17             { $$ = $1 + $2; }
18             | { $$ = 0; }
19             ;
```

```
1  %{
2  #include <iostream>
3  #include <cstring>
4  #include "parse.tab.h"
5  %{
6
7  number     [0-9]+
8
9  %%
10
11 {number}   { yylval = atoi(yytext);
12             return NUMBER;
13           }
14 "\n"       { return CR; }
15 .          { ; }
16
17 %%
18 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.

```
1  %{
2  #include <iostream>
3  extern int yylex();
4  extern int yylval;
5  void yyerror(const char * msg);
6  %}
7  %token CR NUMBER PLUS MINUS MULT DIV
8  %%
9  lines    : lines expr CR
10         { std::cout << "result: " << $2 << std::endl; }
11         | lines CR
12         |
13         ;
14  expr     : expr PLUS term
15         { $$ = $1 + $3; }
16         | expr MINUS term
17         { $$ = $1 - $3; }
18         | term
19         { $$ = $1; }
20         ;
21  term     : term MULT factor
22         { $$ = $1 * $3; }
23         | term DIV factor
24         { $$ = $1 / $3; }
25         | factor
26         { $$ = $1; }
27         ;
28  factor   : NUMBER
29         ;
30  %%
31  void yyerror(const char * msg) { std::cout << msg << std::endl; }
```

```
1  %{
2  #include "parse.tab.h"
3  %}
4
5  %%
6
7  "+"      { return PLUS; }
8  "-"      { return MINUS; }
9  "*"      { return MULT; }
10 "/"      { return DIV; }
11 [0-9]+   { yylval = atoi(yytext);
12           return NUMBER;
13           }
14 "\n"     { return CR; }
15 <<EOF>>  { yyterminate(); }
16
17 %%
18 int yywrap() {
19     yylex_destroy();
20     return 1;
21 }
```

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.

```
1 class WordCount {
2 public:
3     static WordCount* getInstance() {
4         if ( !instance ) instance = new WordCount;
5         return instance;
6     }
7     void incrCount()      { ++count; }
8     int getCount() const { return count; }
9 private:
10     static WordCount* instance;
11     int count;
12     WordCount() : count(0) {}
13 };
```

```
1 %{
2 #include "wordCount.h"
3 %{
4 word      [a-zA-Z]+
5 %x START
6
7 %%
8 \      { BEGIN(START);      }
9 <START>{word} { WordCount::getInstance()->incrCount(); }
10 <START>[""] {      }
11 <START>[""] { BEGIN(INITIAL); }
12
13 {word}      {      }
14 .           {      }
15 "\n"       {      }
16 %%
17 int yywrap() { return 1; }
```

```
1 #include <iostream>
2 #include "wordCount.h"
3 int yylex();
4 WordCount* WordCount::instance = nullptr;
5 int main() {
6     yylex();
7     std::cout << "Number of words in quotes: "
8               << WordCount::getInstance()->getCount()
9               << std::endl;
10 }
```