The following exam is pledged. All answers are to be given on the provided answer sheet. The test is closed book, closed note, and closed calculator. If you believe more than one answer is acceptable, choose the best answer. *YOU MUST HAND IN ALL COPIES OF THE TEST AND THE ANSWER SHEET*. Leaving out or misentering identification information on the answer sheet will cause a 5 point penalty.

On Questions 1 – 14 the following definition and prototypes are in effect. Please observe that the definitions of the `Widget` member functions are very similar to the definitions given in Homework 7. The importance differences are that debugging output statements have been removed and a new member function `concatenate()` has been added to the `Widget` class definition.

```
class Widget {
   public:
      Widget(const string &id = "ID", const string &value = "Value");
      Widget(const Widget &w);
      string getID() const;
      string getValue() const;
      void setValue(const string &s);
      Widget& operator = (const Widget &source);
      void insert(ostream &sout) const;
      void extract(istream &sin);
      Widget concatenate(const Widget &suffix) const;
   protected:
      void setID(const string &s);
   private:
      string thisID;
      string thisValue;
};

istream& operator >> (istream &sin, Widget &object);

ostream& operator << (ostream &sout, Widget &object);

Widget operator + (const Widget &w1, const Widget &w2);
```

In addition, the following definitions are in effect.

```
Widget::Widget(const string &id, const string &value) {
   setID(id);
   setValue(value);
}

string Widget::getID() const {
   return thisID;
}

string Widget::getValue() const {
   return thisValue;
}

void Widget::setID(const string &s) {
   thisID = s;
}

void Widget::setValue(const string &s) {
   thisValue = s;
```

```
}

void Widget::insert(ostream &sout) const {
   sout << "(" << getId() << " - " << getValue() << ")";
}

void Widget::extract(istream &sin) {
   string newValue;
   sin >> newValue;
   setValue(newValue);
}
Widget& Widget::operator = (const Widget &source) {
   string newValue = source.getValue();
   setValue(newValue);
   return *this;
}

istream& operator >> (istream &sin, Widget &object) {
   object.extract(sin);
   return sin;
}

ostream& operator << (ostream &sout, const Widget &object) {
   object.insert(sout);
   return sout;
}

Widget operator + (const Widget &w1, const Widget &w2) {
   return w1.concatenate(w2);
}
```

PART I: Multiple choice questions regarding the Widget library. Each of the following questions should be answered in isolation, i.e., if part of some question causes for a change to a class, function, or variable, that change is not in effect in any other question.

1. How many `Widget` constructors can be used by `Widget` clients?
   (a) 0
   (b) 1
   (c) 2
   (d) 3
   (e) more than 3

2. How many `Widget` inspectors can the `Widget` class designer use?
   (a) 0
   (b) 1
   (c) 2
   (d) 3
   (e) more than 3

3. How many of the following `Widget` member functions are allowed to get the value of `Widget` data member `thisValue`?

```
Widget(const string &id = "ID", const string &value = "Value");
Widget& operator = (const Widget &source);
void insert(ostream &sout) const;
void setID(const string &s);
```

   (a) 0
   (b) 1
   (c) 2
   (d) 3
   (e) 4

4. The source code for `Widget` member function `extract()` is most likely to be found in which of the following files?
   (a) Widget.h
   (b) Widget.cpp
   (c) Widget.obj
   (d) Widget.lib
   (e) ClientCode.cpp

5. The prototype for `Widget` auxiliary operator `<<` is most likely to be found in which of the following files?
   (a) Widget.h
   (b) Widget.cpp
   (c) Widget.obj
   (d) Widget.lib
   (e) ClientCode.cpp

6. Which of the following definitions implements a shallow `Widget` copy constructor?

(a)
```
Widget::Widget(const Widget &w) {
   string id = w.getID();
   string value = w.getValue();
   Widget(id, value);
}
```

(b)
```
Widget::Widget(const Widget &w) {
   string id = w.getID();
   string value = w.getValue();
   setID(id);
   setValue(value);
}
```

(c)
```
Widget::Widget(const Widget &w) {
   thisID = w.thisID;
   thisValue = w.thisValue;
}
```

(d) More than one of the preceding choices with choice b being preferred.

(e) Choices a, b, and c are equally good

In Questions 7 – 9, the following definitions are in effect.

```
const Widget W("W", "abc");
Widget X("X", "def");
Widget Y("Y", "ghi");
Widget Z("Z", "jkl");
```

7. What is the output (if any) of the following code segment?

```
X = Y = Z;
cout << X << endl;
```

(a) `(X - abc)`

(b) `(X - def)`

(c) `(X - jkl)`

(d) `(Z - jkl)`

(e) none of the above

8. How many non-constructors `Widget` member functions can be invoked with `W` as the invoking object?

(a) 0

(b) 1

(c) 2

(d) 3

(e) more than 3

9. In which file(s) is the following statement most likely to occur?

```
X.setID("aaa");
```

(a) Widget.cpp
(b) ClientCode.cpp
(c) ClientCode.exe
(d) Both Widget.cpp and ClientCode.cpp
(e) None of the above

Question 10 – 14 are concerned with implementing `Widget` member function `concatenate()`. The function is to return a `Widget` object whose data member `thisID` is to have value of the left operand's data member `thisID` and whose data member `thisValue` is to be a concatenation of the left and right operand `thisValue` data members. Thus, in particular, we want the function to have a definition that allows the following code segment

```
Widget X("X", "def");
Widget Y("Y", "ghi");
Widget Z("Z", "jkl");
cout << X + Y << endl;
cout << Y + Z << endl;
```

to have output

```
(X - defghi)
(Y - ghijkl)
```

Your implementation is to begin by defining three string variables `id`, `leftValue`, and `rightValue`, where their names imply their use. With those variables you are to define a `Widget result` representing the return value for the function.

10. Which of the following is the appropriate definition for `id`?

(a) `string id = getID();`
(b) `string id << getID();`
(c) `string id << thisID;`
(d) `string id = suffix.getID();`
(e) none of the above

11. Which of the following is the appropriate definition for `leftValue`?

(a) `string leftValue = Widget;`
(b) `string leftValue = getValue();`
(c) `string leftValue << getValue();`
(d) `string leftValue << thisValue;`
(e) none of the above

12. Which of the following is the appropriate definition for `rightValue`?

(a) `string rightValue = suffix;`
(b) `string rightValue = suffix.getValue();`
(c) `string rightValue << suffix.getValue();`
(d) `string rightValue << suffix.thisValue;`
(e) none of the above

13. Which of the following is the appropriate definition for `result`?

(a) `string result = leftValue + rightValue;`

(b) `string result << leftValue + rightValue;`

(c) `Widget result(id, leftValue + rightValue);`

(d) `Widget result << id + leftValue + rightValue;`

(e) none of the above

14. Which of the following is the appropriate expression for the return statement.

(a) `result`

(b) `concatenate = result`

(c) `concatenate.result`

(d) None are necessary as `result` has been already set

(e) None of the above

PART II: Multiple Choice

15. How many of the following statements are valid global function prototypes?

```
a(i);
int b(int i);
int c(int i) const;
int d(int 7);
```

(a) 0

(b) 1

(c) 2

(d) 3

(e) 4

16. Suppose the file output.txt contains the following

```
1 2 3
```

What would be the contents of the file after the following is run?

```
#include <fstream>
using namespace std;
int main() {
   ofstream out("output.txt");
   out << 4 << endl;
   return 0;
}
```

(a) `1 2 3`
`4`

(b) `1 2 34`

(c) Choice a or b depending whether output.txt's last character was an invisible newline character.

(d) 4

(e) Does not compile because `out` is not a valid name for an `ofstream`.

17. What is the output of the following program? .

```
#include <iostream>
using namespace std;
int i = 0;
int I = 1;
int main() {
   int i = 2;
   int I = 3;
   ::i = 10;
   I = 20;
   cout << i << " " << ::i << " " << I << " " << ::I << endl;
   return 0;
}
```

(a) `2 10 20 1`

(b) `2 0 3 1`

(c) `2 2 3 3`

(d) `10 10 20 20`

(e) None of the above

18. Consider the following function `whassup()`

```
void whassup(int &a, int b, int &c) {
   a = 10;
   b = 20;
   c = 30;
}
```

What is the output of the following program fragment?

```
int main() {
   int a = 1;
   int b = 2;
   int c = 3;
   whassup(b, a, c); /* function call */
   cout << a << " " << b << " " << c << endl;
   return 0;
}
```

(a) `1 10 30`

(b) `1 2 3`

(c) `10 20 30`

(d) `10 2 30`

(e) None of the above.

19. In the previous question, if we replaced the line ending with the comment *function call* with

```
whassup(b, b, b);
```

What would the output of the program fragment then be?

(a) `10 20 30`
(b) `10 2 30`
(c) `2 2 2`
(d) `1 30 3`
(e) None of the above.

PART III True and False True = red**, False = black**

20. A header file is a collection of function interfaces, constant and variable object definitions, and class descriptions. The header file is incorporated into the program through an include directive.

21. When a function is invoked, flow of control is transferred from the invoking function to the invoked function. When the invoked function completes, control is transferred back to the invoking function. If the invoked function returns a value, then that value is essentially substituted for the invocation.

22. Every function invocation creates an activation record. The values of the formal parameters and other objects defined in the function are kept in the activation record.

Questions 23 – 27 consider the following program.

```
#include <iostream>
using namespace std;

double f(double x) {
   x = 2 * x;
   return 2*x;
}

int main() {
   cout << "Enter number: ";
   double number;
   cin >> number;
   cout << f(number) << endl;
}
```

23. Variable `number` is an actual parameter.

24. Variable `number` is a formal paramter.

25. Variable `x` is an actual parameter.

26. Variable `x` is a value parameter.

27. Variable `number` is changed in the invocation `f(number)`.

28. When passing an object to a function, if that object is not one of the built-in C++ types (e.g., `char`) then it is good programming practice to pass it by value when we know that this parameter will not be altered by the function.

29. Output streams can be passed by value.

For Questions 30 – 34, suppose there is a class `C` that has a member function `doIt()` and `makeIt()` with prototypes:

```
void C::doIt(C &x);
void C::makeIt(const C &x) const;
```

30. A type `C` constant can invoke member function `doIt()`.
31. A type `C` non-constant can invoke member function `doIt()`.
32. A type `C` constant can be passed as a parameter to member function `doIt()`.
33. A type `C` constant can invoke member function `makeIt()`.
34. A type `C` non-constant can be passed as a parameter to member function `makeIt()`.

For Questions 35 – 39, the following function prototype and global variable definitions are in effect.

```
void f(double &v);
int count = 1;
double val = 1.0;
const double nbr = 1.0;
```

35. Changes to `v` within the function `f()` affect the actual parameter passed in the function call.
36. The invocation `f(count)` is a legal invocation.
37. The invocation `f(val)` is a legal invocation.
38. The invocation `f(nbr)` is a legal invocation.
39. The invocation `f(3.5)` is a legal invocation.

For Questions 40 – 41, we are concerned with a function `PromptAndGet()` which has two parameters: an integer extracted from a stream and the stream to use for the extraction. We first write this prototype as

```
void PromptAndGet(int val, istream sin);
```

but we realize that it's wrong. We decide at least one of these parameters must have an ampersand symbol '&' before the parameter name.

40. Parameter `val` should get an ampersand.
41. Parameter `sin` should get an ampersand.

For Questions 42 – 43, we are concerned with a function `MoveRect()` with the following definition.

```
void MoveRect(const RectangleShape &R1) {
   R1.SetPosition(1.0, 1.0);
}
```

42. The definition of `MoveRect()` generates a compiler error.
43. Function `MoveRect()` compiles and when run `RectangleShape` object `R1`'s position data members will be updated only after `MoveRect()` returns.

For Questions 44 – 47, we are concerned with the following program.

```
#include <iostream>
using namespace std;
void f(double x, double y) {
   cout << "f double double! ";
}
void f(double x, int i) {
   cout << "f double int! ";
}
int main() {
   f(1.1, 3.3);
   f(2.2, 3);
   return 0;
}
```

44. The program will not successfully compile.

45. The program prints: *f double double! f double double!*

46. The program prints: *f double double!* but then it causes a run-time error.

47. The program prints: *f double double! f double int!*

For Questions 48 – 50, suppose you have been asked to develop a single function that can be called in the following three ways:

```
Funk(99, "==", 2.7);
Funk(7, "++");
Funk(1);
```

48. It is not possible to write one function that can be used in three different ways like this.

49. The function prototype could be:

```
void Funk(int i, const string &s = "$$", double x);
```

50. The function prototype could be:

```
void Funk(int i = -1, const string &s = "$$", double x = 3.14);
```