**Instructions:** This homework assignment covers some of the basic C++ background you should have in order to take this course.

You will submit your answers to the Curator system.

---

I. **Pointers and Memory Management**

For questions 1 through 6, assume the variable declarations and initial memory layout shown below:

```cpp
int a = 42, b = 17;
int *p, *q;
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6AFDF4</td>
</tr>
<tr>
<td>b</td>
<td>6AFDF0</td>
</tr>
<tr>
<td>p</td>
<td>6AFDEC</td>
</tr>
<tr>
<td>q</td>
<td>6AFDE8</td>
</tr>
</tbody>
</table>

Suppose the following statements are executed:

```cpp
p = &b;  // statement 1
q = &a;  //           2
*p = *q;  //           3
q = p;   //           4
```

For questions 1 through 6, chose from the following answers:

1) 6AFDF4
2) 6AFDF0
3) 6AFDEC
4) 6AFDE8
5) 17
6) 42
7) Unknown
8) None of these

1. Immediately after the execution of statement 1, what is the value of: `p`
2. Immediately after the execution of statement 2, what is the value of: `*q`
3. Immediately after the execution of statement 2, what is the value of: `&q`
4. Immediately after the execution of statement 3, what is the value of: `p`
5. Immediately after the execution of statement 3, what is the value of: `*p`
6. Immediately after the execution of statement 4, what is the value of: `*q`

---

7. Assume the variable declarations:

```cpp
int *Foo = new int(0);
```

Which of the following statements will increment the target of `Foo`?

1) `Foo++`;
2) `(*Foo)++`;
3) `*Foo++`;
4) 2 and 3 only
5) None of these
8. Both code fragments below will compile but the one on the right contains a logic error, and will probably cause a runtime error. Why?

```
int x = 5;
int *p = new int(x);
delete p;
```

```
int x = 5;
int *p = &x;
delete p;
```

1) Assigns an address to an int variable.
2) Assigns an int value to a pointer.
3) Attempts to delete a statically allocated variable.
4) None of these

For questions 9 through 11 assume that we have a dynamically allocated array A of integers of dimension Size, with memory layout as shown:

```
const int Size = 5;
int *A = new int[Size];
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7D0E80</td>
</tr>
<tr>
<td>1</td>
<td>7D0E7C</td>
</tr>
<tr>
<td>2</td>
<td>7D0E78</td>
</tr>
<tr>
<td>3</td>
<td>7D0E74</td>
</tr>
<tr>
<td>4</td>
<td>7D0E70</td>
</tr>
</tbody>
</table>

9. Which code fragment(s) could be inserted in the blank in order to safely initialize each element of A to zero?

```
int* p = &A[0];
for (int Idx = 0; Idx < Size; Idx++, p++) {
    [ ];
}
```

1) A[Idx] = 0;
2) *p = 0;
3) *A = 0;
4) *Idx = 0;
5) All of the above
6) 1 and 2 only
7) 1 and 3 only
8) 1 and 4 only
9) None of these

10. What value will be printed by the code fragment:

```
for (int Idx = 0; Idx < Size; Idx++) {
    A[Idx] = int(&A[Idx]); // typecast converts to int
}
```

1) 7D0E70
2) 7D0E74
3) 7D0E78
4) 7D0E7C
5) 7D0E80
6) Unknown
7) None of these

11. Assuming only the initial declarations given above, what logical error(s) would result if the following statement were executed: A = new int[2*Size];

1) A dangling pointer would result (a pointer whose value is the address of memory that the program no longer owns).
2) A memory leak would result (the program would own memory that it could no longer access).
3) Both a dangling pointer and a memory leak would result.
4) Neither 1 nor 2, but some other logical error would result.
5) No logical error would result.
Consider implementing a function to dynamically-allocate an array of integers and set all its elements to zero:

```cpp
void ZeroIt(A, const int Size) { // Line 1
    A = _________; // Line 2
    int Idx;
    for (Idx = 0; Idx < Size; Idx++) {
        A[Idx] = 0;
    }
}
```

The function is to be called from `main()`, which will then pass the array to other functions for their use.

12. What would be the best way to fill the blank in Line 1?

1) `int*&` 
2) `int*` 
3) `const int*` 
4) `int* const` 
5) `const int* const` 
6) All of these are bad

13. What would be the best way to fill the blank in Line 2?

1) `int[Size]` 
2) `int*[Size]` 
3) `new int[Size]` 
4) `new int*[Size]` 
5) `new int` 
6) All of these are bad

Consider the following short program, which was tested under Windows 2000:

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

string* getString();
void initString(string* S);

int main() {
    string *q, *r; // Line 1
    q = getString(); // Line 2
    cout << *q << endl; // Line 3
    initString(r); // Line 4
    cout << *r << endl; // Line 5
    return 0;
}

string* getString() { // Line 6
    string p = "ha ha"; // Line 7
    return &p; // Line 8
}

void initString(string* S) { // Line 9
    string Word = "ha"; // Line 10
    int Count;
    for (Count = 0; Count < 1000; Count++)
        *S = *S + Word; // Line 11
}
```

14. The execution of the statement Line 3 did not produce the output `ha ha`. Precisely identify the logic error that causes this. (Choose only one response.)

1) The variable `q` in `main` should not be declared as a pointer.
2) The assignment in Line 2 suffers from a shallow copy problem.
3) The statement in Line 2 should be: `*q = getString();`
4) The statement in Line 3 should be: `cout << *q << endl;`
5) The statement in Line 3 should be: `cout << &q << endl;`
6) The return type of the function in Line 6 should not be a pointer.
7) The initialization of the `string` variable in Line 7 is incorrect.
8) The statement in Line 8 should be: `return p;`
9) The statement in Line 8 should be: `return *p;`
15. The execution of the statement Line 4 caused the following runtime error message:

```
The instruction at "0x00401ae0" referenced memory at "0xcddddddc". The memory could not be written.
```

Explain carefully what directly causes this runtime error. (Note that this is not directly caused by the logic error referred to in the previous question.)

1) The variable \( r \) in \texttt{main} should not be declared as a pointer.
2) The statement in Line 4 should be: \texttt{initString(*r);}
3) A target should be allocated for \( r \) before the call to \texttt{initString()} is made.
4) The function parameter in Line 9 should be passed by reference.
5) The initialization of the \texttt{string} variable in Line 10 is incorrect.
6) The body of the for loop in the function \texttt{initString()} executes too many times; a \texttt{string} variable cannot hold that much data.
7) The statement in Line 11 should be: \( S = S + \text{Word}; \)
8) The statement in Line 11 should be: \( &S = &S + \text{Word}; \)

```cpp
#include <fstream>
#include <string>
using namespace std;

string* getDataLine(ifstream& In) {
    string *p = new string;
    getline(In, *p, '\n');
    return p;
}

void toUpper(string* S) {
    int Idx;
    for (Idx = 0; Idx < S->length(); Idx++)
        cout << char( toupper(*S)[Idx] ) ;
    cout << endl;
}

int main() {
    ifstream In("Data.txt");
    string *q;
    q = getDataLine(In);
    while ( In ) {
        toUpper(q);
        q = getDataLine(In);
    }
}
```

16. The program given above executes correctly, in the sense that it produces correct output. However, the program does contain a serious logical flaw relating to memory management. What is the flaw?

1) A memory leak occurs; i.e., the program allocates memory dynamically and never deallocates that memory.
2) A dangling pointer occurs; i.e., a pointer stores the address of memory that has been deallocated.
3) A loop bound is incorrect, causing an array index to go slightly out of bounds.
4) None of these.

17. Where does the error occur?

1) Within the loop in \texttt{main}.
2) After the end of the loop in \texttt{main}.
3) Within the function \texttt{getDataLine()}.
4) Within the function \texttt{toUpper()}.
5) None of these.
II. Recursion

18. Given the recursive function below, what is the value of the expression \( \text{Sum}(5) \)?

```c
int Sum( int n ) {
    if ( n < 8 )
        return ( n + Sum(n + 1) );
    else
        return 2;
}
```

1) 5  
2) 13  
3) 20  
4) 28  
5) None--the result is infinite recursion  
6) None of these

For questions 19 through 23, consider the following function, `countEm()`, and the associated recursive helper function `rCount()`, which are intended to determine how many times a specified `Value` occurs in a given array holding `Size` elements:

```c
int countEm(int Value, const int Array[], int Size) {
    return rCount(Value, Array, Size);
}

int rCount(int Value, const int Array[], int Size) {
    if (Size <=    // Line 1
        return    // Line 2
    else if (Array[    // Line 3
        return    // Line 4
    else
        return    // Line 5
}
```

19. How should the blank in Line 1 be filled?

1) 0  
2) `Size`  
3) `-1`  
4) `Value`  
5) `INT_MAX`  
6) None of these

20. How should the blank in Line 2 be filled?

1) 0  
2) 1  
3) `Size`  
4) `INT_MAX`  
5) `Value`  
6) None of these

21. How should the blank in Line 3 be filled?

1) `Size`  
2) `Size - 1`  
3) 0  
4) 1  
5) `INT_MAX`  
6) None of these

22. How should the blank in Line 4 be filled?

1) `rCount(Value, Array, Size - 1)`  
2) `rCount(Value, Array, Size)`  
3) `1 + rCount(Value, Array, Size - 1)`  
4) `1 + rCount(Value, Array, Size)`  
5) 1  
6) None of these
23. How should the blank in Line 5 be filled?

   1) rCount(Value, Array, Size - 1)
   2) rCount(Value, Array, Size)
   3) 1 + rCount(Value, Array, Size - 1)
   4) 1 + rCount(Value, Array, Size)
   5) 1
   6) None of these

III. Lists

For questions 24 through 28, consider the following declaration and implementation for a circular array-based Queue.

**Note:** In this implementation the `Front` pointer is the index of the first element and the `Rear` pointer is the index of the next available cell (not of the last element in the Queue).

```cpp
class Queue {
private:
    int Size;                      // dimension of queue array
    int Front;                     // index for next deletion
    int Rear;                      // index for next insertion
    Item *Q;                        // queue array
public:
    Queue(int Sz);                      // deep copy constructor, and
    Queue(const Queue& Source);    // assignment overload
    bool Enqueue(Item It);         // insert Item at Rear; return false if fails
    bool Dequeue(Item& It);        // delete Item at Front and return it; return
                                      //    false if fails
    bool isEmpty() const;          // returns true if empty
    bool isFull() const;           // return false if full
    void Display();                // prints queue contents to cout
    int getSize() const;           // returns size of queue
    void Clear();                  // resets queue to empty state (no deallocation)
~Queue();
};

Queue::Queue(int Sz) {
    Size = Sz;
    Q = new Item[Sz];
    if (Q == NULL)
        Size = 0;
    Front = 0;
    Rear = 0;
}

bool Queue::isEmpty() {
    return (Rear == Front);
}

bool Queue::isFull() {
    return (Front == ________); // Line 1
}

void Queue::Clear() {
    Front = Rear = 0;
}

bool Queue::Enqueue(Item It) {
    if ( isFull() )
        return false;
    _____________ = It;   // Line 2
    Rear = ____________;   // Line 3
    return true;
}

bool Queue::Dequeue(Item& It) {
    if ( isEmpty() )
        return false;
    It = ____________;  // Line 4
    Front = ____________; // Line 5
    return true;
}

Queue::~Queue() {
    delete [] Q;
}
24. How should the blank in Line 1 be filled? (Remember that the states “full” and “empty” are different.)

1) \((\text{Rear} + 1) \mod \text{Size}\)  
2) \(\text{Size} - \text{Rear}\)  
3) \(\text{Rear}\)  
4) \(\text{Rear} + 1\)  
5) None of these

25. How should the blank in Line 2 be filled?

1) \(\text{Q}[\text{Front}]\)  
2) \(\text{Q}[\text{Size}]\)  
3) \(\text{Q}[\text{Rear}]\)  
4) \(\text{Q}[\text{Rear} + 1]\)  
5) None of these

26. How should the blank in Line 3 be filled?

1) \(\text{Rear} - 1\)  
2) \(\text{Rear} + 1\)  
3) \(\text{Front}\)  
4) \((\text{Rear} + 1) \mod \text{Size}\)  
5) None of these

27. How should the blank in Line 4 be filled?

1) \(\text{Q}[\text{Front} - 1]\)  
2) \(\text{Q}[\text{Front} + 1]\)  
3) \(\text{Q}[\text{Rear}]\)  
4) \(\text{Q}[\text{Rear} + 1]\)  
5) None of these

28. How should the blank in Line 5 be filled?

1) \(\text{Front} - 1\)  
2) \((\text{Front} - 1) \mod \text{Size}\)  
3) \(\text{Front} + 1\)  
4) \((\text{Front} + 1) \mod \text{Size}\)  
5) None of these

For questions 29 through 33 we consider implementing a function (NOT a member function) to sort the entries of a Queue object into ascending order, where Queue is declared on page 2. Given the shell below, which is a variant of selection sort, and assuming that int has been typedef’d to Item:

```cpp
void Sort(Queue& Q) {
    int numItems = Q.getSize();
    int SmallSoFar, Look;

    Queue Temp(numItems); // 1: holds unsorted values from Q
    Queue Sorted(numItems); // 2: holds sorted values

    while ( !Q.isEmpty() ) {
        while ( !Q.isEmpty() ) { // 3: first is smallest seen yet
            Q.Dequeue(Look); // 4: grab next element from Q
            if (Look < SmallSoFar) { // 5: if it's a new minimum
                SmallSoFar = Look; // 6: save old minimum for future
            } else // not
                SmallSoFar = Look; // 8: save current value for future
        }
        Sorted.Enqueue(SmallSoFar); // 9: put smallest in results queue
        Q = Sorted; // 12: put sorted values into Q
    }
```
29. How should the blank in line 3 be filled?

1) SmallSoFar = 0  
2) SmallSoFar = Q[0]  
3) Q.Dequeue(Look)  
4) Q.Dequeue(SmallSoFar)  
5) None of these

30. How should the blank in line 6 be filled?

1) It should be blank  
2) Q.Enqueue(SmallSoFar)  
3) Temp.Enqueue(SmallSoFar)  
4) Temp.Enqueue(Look)  
5) None of these

31. How should the blank in line 8 be filled?

1) Temp.Enqueue(SmallSoFar)  
2) Temp.Enqueue(Look)  
3) Q.Enqueue(SmallSoFar)  
4) Q.Enqueue(Look)  
5) None of these

32. How should the blank in line 10 be filled?

1) Temp = Q  
2) Q = Sorted  
3) Q = Temp  
4) Sorted = Temp  
5) None of these

33. How should the blank in line 11 be filled?

1) Temp = NULL  
2) delete Temp  
3) Temp.Clear()  
4) Sorted.Clear()  
5) None of these

IV. Inheritance and Polymorphism

For questions 34 through 37, assume that Foo and Bar are C++ classes, and that the class Foo is derived, using public inheritance, from the class Bar.

34. If X is an object of type Bar, then the member functions of X:

1) can directly access only the added public members of class Foo.  
2) cannot directly access any of the added members of class Foo.  
3) can directly access the added public, protected, and private members of class Foo.  
4) can directly access only the added public and protected members of class Foo.  
5) None of these

35. Suppose that an object X of type Foo is declared. Then:

1) a constructor for the base class, Bar, will be executed after any constructor for the derived class, Foo.  
2) no constructor for the base class, Bar, will be executed at all.  
3) a constructor for the base class, Bar, will be executed before any constructor for the derived class, Foo.  
4) constructors for both classes will be executed at the same time.  
5) a constructor for the base class, Bar, may (or may not) be executed, and that may take place either before or after the execution of a constructor for the derived class, Foo.  
6) None of these
36. Suppose that an object \( X \) of type \( \text{Foo} \) is declared. Then when the lifetime of that object ends:

1) destructors for both classes will be executed at the same time.
2) the destructor for the base class, \( \text{Bar} \), will be executed after the destructor for the derived class, \( \text{Foo} \).
3) the destructor for the base class, \( \text{Bar} \), will not be executed at all.
4) the destructor for the base class, \( \text{Bar} \), will be executed before the destructor for the derived class, \( \text{Foo} \).
5) the destructor for the base class, \( \text{Bar} \), may (or may not) be executed, and that may take place either before or after the execution of the destructor for the derived class, \( \text{Foo} \).
6) None of these

37. Which of the following is true?

1) Public members of \( \text{Foo} \) become public members of \( \text{Bar} \).
2) Public members of \( \text{Bar} \) become public members of \( \text{Foo} \).
3) Public members of \( \text{Foo} \) become private members of \( \text{Bar} \).
4) Public members of \( \text{Bar} \) become private members of \( \text{Foo} \).
5) 2 and 3 only
6) None of these

For questions 38 and 39, suppose that a C++ class \( D \) is derived from a base class \( B \), that class \( B \) has a public member function \( F() \), and class \( D \) redefines its own version of \( F() \). At execution time, suppose that a pointer to a \( D \) object is passed to the following function:

```cpp
void Foo(B* x) {
    x->F();
}
```

38. If \( F() \) is declared to be virtual in class \( B \), whose version of \( F() \) is called?

1) class \( D \)'s version
2) class \( B \)'s version
3) Both versions are called.
4) Neither version is called.

39. If \( F() \) is not declared to be virtual in class \( B \), whose version of \( F() \) is called?

1) class \( D \)'s version
2) class \( B \)'s version
3) Both versions are called.
4) Neither version is called.

For questions 40 and 41, suppose that a C++ class \( D \) is derived from a base class \( B \), that class \( B \) has a public member function \( F() \), and class \( D \) redefines its own version of \( F() \). At execution time, suppose that a \( D \) object is passed to the following function:

```cpp
void Foo(B x) {
    x.F();
}
```

40. If \( F() \) is declared to be virtual in class \( B \), whose version of \( F() \) is called?

1) class \( D \)'s version
2) class \( B \)'s version
3) Both versions are called.
4) Neither version is called.
41. If $F()$ is not declared to be virtual in class B, whose version of $F()$ is called?

1) class D's version
2) class B's version
3) Both versions are called.
4) Neither version is called.

For questions 42 through 45, assume the following class declarations:

```cpp
class Base {
public:
    virtual void F();
    virtual void G() = 0;
    void H();
};

class D : public Base {
public:
    void F();
    void G();
    void H();
};

class E : public D {
public:
    void F();
    void G();
};
```

Suppose corresponding implementations are given for each class, and consider the following `main()`:

```cpp
int main() {
    D* pD = new D;
    Base* pB = (Base*) pD;

    pB->F(); // call 1
    pB->G(); // call 2
    pB->H(); // call 3

    E* pE = new E;
    pB = (Base*) pE;
    pD = (D*) pE;

    pB->F(); // call 4
    pD->F(); // call 5
    pB->G(); // call 6
    pE->H(); // call 7
    return 0;
}
```

42. Which function is called in the statement labeled `call 1`?

1) Base::F()
2) D::F()
3) E::F()
4) Two or more of the above.
5) None of these.

43. Which function is called in the statement labeled `call 2`?

1) Base::G()
2) D::G()
3) E::G()
4) Two or more of the above.
5) None of these.
44. Which function is called in the statement labeled call 5?

1) Base::F()  
2) D::F()  
3) E::F()  
4) Two or more of the above.  
5) None of these.

45. Which function is called in the statement labeled call 7?

1) Base::H()  
2) D::H()  
3) E::H()  
4) Two or more of the above.  
5) None of these.

V. Other Class Issues

For questions 46 through 50, consider the classes Track and Album:

```cpp
class Track {
private:
    string Title;
    int    Length;
public:
    Track(string T, int L);
    string getTitle() const;
    int    getLength() const;
};

class Album {
private:
    string Title;
    string Artist;
    int    numTracks;
    Track* PlayList;
public:
    Album(string T, string A, int nT) {
        Title = T;
        Artist = A;
        numTracks = nT;
        PlayList = new Track[nT];
    }
    bool AddTrack(const Track& T);
    Track getTrack(int Position) const;
    int getNumTracks() const;
    ~Album();
}

Consider the following non-member function, which computes the sum of the lengths of all the tracks on an album:

```cpp
int Length(Album CD) {
    int totalLength = 0;
    int Idx;
    for (Idx = 0; Idx < CD.getNumTracks(); Idx++) {
        totalLength += CD.getTrack(Idx).getLength();
    }
    return totalLength;
}
```
46. The call Length(myAlbum) will have an unfortunate side effect (even though the body of the function is correct). What is that effect?

1) myAlbum.numTracks is changed.
2) The array of tracks, myAlbum.PlayList[], is deleted.
3) The destructor for myAlbum is invoked.
4) All of the above.
5) 2 and 3 only
6) None of these

47. Assuming that myAlbum is declared in the calling function, will the call Length(myAlbum) also cause a runtime exception at the end of the calling function?

1) Perhaps yes, perhaps no, depending on factors not specified in the question.
2) Yes, definitely.
3) No, definitely not.

48. Given that the interface and implementation of Length() cannot be changed, and that the use of global variables is unacceptable, which of the following actions is/are necessary to eliminate the difficulties cited in questions 46 and 47?

1) A deep assignment operator overload should be implemented for the class Album.
2) The destructor for the class Album should be removed from the class.
3) A deep copy constructor should be implemented for the class Album.
4) All of the above.
5) 1 and 2 only
6) None of these

49. Aside from the action(s) you chose in question 48, which of those actions should also be done in order to produce a robust, general implementation?

1) A deep assignment operator overload should be implemented for the class Album.
2) The destructor for the class Album should be removed from the class.
3) A deep copy constructor should be implemented for the class Album.
4) All of the above.
5) 1 and 2 only
6) None of these

50. What feature of the class Album or of the class Track should have immediately tipped the implementer of that class to take the actions referred to in questions 48 and 49?

1) Track does not implement a destructor.
2) Album contains an aggregation of Track objects.
3) Album contains a pointer to dynamically allocated storage.
4) Track contains a string object.
5) None of these