

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

Opscan forms will be passed out in class. Write your name and code your ID number on the Opscan form. Turn in your completed Opscan at the appropriate time listed below. Opscans will not be accepted at any other time.

I. Pointers

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

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

Identifier	Address
a	006AFDF4
b	006AFDF0
p	006AFDEC
q	006AFDE8

Suppose the following statements are executed:

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

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

- | | | |
|-------------|-------------|------------------|
| 1) 006AFDF4 | 4) 006AFDE8 | 7) Unknown |
| 2) 006AFDF0 | 5) 17 | 8) None of these |
| 3) 006AFDEC | 6) 42 | |

- Immediately after the execution of statement 1, what is the value of: p
- Immediately after the execution of statement 2, what is the value of: *q
- Immediately after the execution of statement 2, what is the value of: &q
- Immediately after the execution of statement 3, what is the value of: p
- Immediately after the execution of statement 3, what is the value of: *p
- Immediately after the execution of statement 4, what is the value of: *q

7. Assume the variable declarations:

```
int Foo = 0;
int *ptr = &Foo;
```

Which of the following statements will increment Foo?

- | | | |
|-----------|--------------|------------------|
| 1) Foo++; | 3) (*Foo)++; | 5) 1 and 4 only |
| 2) ptr++; | 4) (*ptr)++; | 6) None of these |

8. Both code fragments below will compile but the one on the right will 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 variable to a pointer.
 3) Deletes 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];
```

Index	Address
0	007D0E70
1	007D0E74
2	007D0E78
3	007D0E7C
4	007D0E80

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) *A = 0;
 3) *p = 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
}
cout << hex << A[3] << endl; // manipulator causes hex output
```

- 1) 007D0E70
 2) 007D0E74
 3) 007D0E78
 4) 007D0E7C
 5) 007D0E80
 6) Unknown
 7) None of these

11. Assuming only the initial declarations given above, and the code in questions 9, 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.

12. Consider implementing a function to dynamically-allocate an array of integers and reset all its elements to zero:

```
void ZeroIt(_____ A, const int Size) {
    A = new int[Size];
    for (int Idx = 0; Idx < Size; Idx++) {
        A[Idx] = 0;
    }
}
```

Which of the following choices for the blank preceding the formal parameter A is best?

- | | | |
|----------|---------------|---------------------|
| 1) int* | 3) const int* | 5) const int* const |
| 2) int*& | 4) int* const | 6) All of the above |

For questions 13 and 14, assume the declarations:

```
struct Node {
    float Volume;
    Node* Next;
};
Node* headPtr;
```

Also assume that headPtr is the head pointer to a linked list of many Nodes.

13. Which statement renders the head node (ONLY the head node) inaccessible?

- 1) headPtr = headPtr->Next->Next;
- 2) *headPtr = headPtr->Next;
- 3) headPtr->Next = headPtr->Next->Next;
- 4) headPtr->>(*Next) = headPtr->Next->Next;
- 5) None of these

14. Which statement changes the data element of the second node in the list?

- 1) headPtr->Next = 42;
- 2) headPtr->Next->Volume = 42;
- 3) *(headPtr->Next->Volume) = 42;
- 4) *(headPtr->Next) = 42;
- 5) None of these

II. Recursion

15. Given the recursive function below, what is the value of the expression Sum(5) ?

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

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

For questions 16 through 20, consider the following function, `isThere()`, and the associated helper function `ValueInList()`, which are intended to indicate whether a specified `Value` occurs in a given array holding `Size` elements:

```
bool isThere(int Value, const int Array[], int Size) {  
    return ValueInList(Value, Array, Size);  
}  
  
bool ValueInList(int Value, const int Array[], int Size) {  
    if (Size <= _____) // Line 1  
        return _____; // Line 2  
    else if (Array[Size-1] == _____) // Line 3  
        return _____; // Line 4  
    else  
        return ValueInList(Value, Array, _____); // Line 5  
}
```

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

- | | | |
|----------|----------|------------------|
| 1) false | 3) 0 | 5) -1 |
| 2) true | 4) Value | 6) None of these |

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

- | | | |
|----------|-------------|------------------|
| 1) false | 3) Size++ | 5) Value |
| 2) true | 4) Size - 1 | 6) None of these |

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

- | | | |
|----------|-------------|------------------|
| 1) Value | 3) Array[0] | 5) Array[Size-1] |
| 2) true | 4) false | 6) None of these |

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

- | | | |
|----------|-----------|------------------|
| 1) false | 3) Value | 5) Array[Size-1] |
| 2) true | 4) Size-1 | 6) None of these |

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

- | | | |
|----------|-------------|------------------|
| 1) false | 3) Size++ | 5) Value |
| 2) true | 4) Size - 1 | 6) None of these |

III. Lists

For questions 21 through 25, 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).

```
class Queue {
private:
    int Size;           // dimension of queue array
    int Front;         // index for next deletion
    int Rear;          // index for next insertion
    ItemType *Q;       // queue array
public:
    Queue(int Sz);
    Queue(const Queue& Source); // deep copy constructor, and
    Queue& operator=(const Queue& Source); // assignment overload
    bool Enqueue(ItemType Item); // insert Item at Rear; return false if fails
    bool Dequeue(ItemType& Item); // delete Item at Front and return it; return
                                   // false if fails

    bool isEmpty() const;
    bool isFull() const;
    void Display(); // prints queue contents to cout
    int getSize() const; // returns size of queue
    void Clear(); // resets queue to empty state (no deallocation)
    ~Queue();
};
```

<pre>Queue::Queue(int Sz) { Size = Sz; Q = new ItemType[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; }</pre>	<pre>bool Queue::Enqueue(ItemType Item) { if (isFull()) return false; _____ = Item; // Line 2 Rear = _____; // Line 3 return true; } bool Queue::Dequeue(ItemType& Item) { if (isEmpty()) return false; Item = _____; // Line 4 Front = _____; // Line 5 return true; } Queue::~~Queue() { delete [] Q; }</pre>
--	---

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

- | | | |
|-------------|----------------------|------------------|
| 1) Rear | 3) (Rear + 1) % Size | 5) None of these |
| 2) Rear + 1 | 4) Size - Rear | |

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

- | | | |
|----------------|-------------|------------------|
| 1) Q[Rear] | 3) Q[Front] | 5) None of these |
| 2) Q[Rear + 1] | 4) Q[Size] | |

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

- | | | |
|-------------|----------------------|------------------|
| 1) Rear + 1 | 3) (Rear + 1) % Size | 5) None of these |
| 2) Rear - 1 | 4) Front | |

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

- | | | |
|-----------------|----------------|------------------|
| 1) Q[Front + 1] | 3) Q[Rear] | 5) None of these |
| 2) Q[Front] | 4) Q[Rear - 1] | |

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

- | | | |
|-----------------------|-----------------------|------------------|
| 1) Front + 1 | 3) Front - 1 | 5) None of these |
| 2) (Front + 1) % Size | 4) (Front - 1) % Size | |

For questions 26 through 30 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 ItemType:

```

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() ) {
        _____;                // 3: first is smallest seen yet
        while ( !Q.isEmpty() ) {
            Q.Dequeue(Look);        // 4: grab next element from Q
            if (Look < SmallSoFar) { // 5: if it's a new minimum
                _____;        // 6: save old minimum for future
                SmallSoFar = Look;  // 7: reset minimum
            }
            else                    // if not
                _____;        // 8: save current value for future
        }
        Sorted.Enqueue(SmallSoFar); // 9: put smallest in results queue
        _____;                //10: restore unsorted values to Q
        _____;                //11: clear temporary queue
    }
    Q = Sorted;                    //12: put sorted values into Q
}

```

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

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

27. How should the blank in line 6 be filled?
- 1) It should be blank
 - 2) `Q.Enqueue(SmallSoFar)`
 - 3) `Temp.Enqueue(Look)`
 - 4) `Temp.Enqueue(SmallSoFar)`
 - 5) None of these
28. How should the blank in line 8 be filled?
- 1) `Temp.Enqueue(Look)`
 - 2) `Look = Temp.Dequeue()`
 - 3) `Q.Enqueue(Look)`
 - 4) `Q.Enqueue(SmallSoFar)`
 - 5) None of these
29. 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
30. How should the blank in line 11 be filled?
- 1) `Temp = NULL`
 - 2) `delete Temp`
 - 3) `Sorted.Clear()`
 - 4) `Temp.Clear()`
 - 5) None of these
-

IV. Inheritance and Polymorphism

For questions 31 through 34, assume that `Foo` and `Bar` are C++ classes, and that the class `Foo` is derived, using public inheritance, from the class `Bar`.

31. If `X` is an object of type `Bar`, then the member functions of `X`:
- 1) cannot directly access any of the added members of class `Foo`.
 - 2) can directly access only the added public members of class `Foo`.
 - 3) can directly access only the added public and protected members of class `Foo`.
 - 4) can directly access the added public, protected, and private members of class `Foo`.
 - 5) None of these
32. Suppose that an object `X` of type `Foo` is declared. Then:
- 1) no constructor for the base class, `Bar`, will be executed at all.
 - 2) a constructor for the base class, `Bar`, will be executed before any constructor for the derived class, `Foo`.
 - 3) a constructor for the base class, `Bar`, will be executed after any constructor for the derived class, `Foo`.
 - 4) 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`.
 - 5) constructors for both classes will be executed at the same time.
 - 6) None of these

33. Suppose that an object `X` of type `Foo` is declared. Then when the lifetime of that object ends:
- 1) the destructor for the base class, `Bar`, will not be executed at all.
 - 2) the destructor for the base class, `Bar`, will be executed before the destructor for the derived class, `Foo`.
 - 3) the destructor for the base class, `Bar`, will be executed after the destructor for the derived class, `Foo`.
 - 4) the destructor for the base class, `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, `Foo`.
 - 5) destructors for both classes will be executed at the same time.
 - 6) None of these
34. Which of the following is true?
- 1) Public members of `Foo` become public members of `Bar`.
 - 2) Public members of `Foo` become private members of `Bar`.
 - 3) Public members of `Bar` become public members of `Foo`.
 - 4) Public members of `Bar` become private members of `Foo`.
 - 5) 2 and 3 only
 - 6) None of these

For questions 35 and 36, 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:

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

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

For questions 37 and 38, 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:

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

37. If `F()` is declared to be virtual in class `B`, whose version of `F()` is called?
- 1) class `B`'s version
 - 2) class `D`'s version
 - 3) Both versions are called.
 - 4) Neither version is called.

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

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

For questions 39 through 45, assume the following class declarations:

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

```
void 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
}
```

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

- | | | |
|---------------------------|------------------------------|-------------------|
| 1) <code>Base::F()</code> | 3) <code>E::F()</code> | 5) None of these. |
| 2) <code>D::F()</code> | 4) Two or more of the above. | |

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

- | | | |
|---------------------------|------------------------------|-------------------|
| 1) <code>Base::G()</code> | 3) <code>E::G()</code> | 5) None of these. |
| 2) <code>D::G()</code> | 4) Two or more of the above. | |

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

```
int Length(Album CD) {  
    int totalLength = 0;  
  
    for (int 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 destructor for `myAlbum` is invoked.
 - 3) The array of tracks, `myAlbum.PlayList[]`, is deleted.
 - 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) Yes, definitely.
 - 2) No, definitely not.
 - 3) Perhaps yes, perhaps no, depending on factors not specified in the question.
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 copy constructor should be implemented for the class `Album`.
 - 2) A deep assignment operator overload should be implemented for the class `Album`.
 - 3) The destructor for the class `Album` should be removed from the class.
 - 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 solid implementation?
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) `Album` contains an aggregation of `Track` objects.
 - 2) `Album` contains a pointer to dynamically allocated storage.
 - 3) `Track` does not implement a destructor.
 - 4) `Track` contains a string object.
 - 5) None of these