
***READ THIS NOW!***

Failure to read and follow the instructions below may result in severe penalties.

- Print your name in the space provided below.
- Print your name and ID number on the Opscan form and code your ID number correctly on the Opscan form.
- Choose the single best answer for each question — some answers may be partially correct. If you mark more than one answer to a question, you will receive no credit for any of them.
- Unless a question involves determining whether given C++ code is syntactically correct, assume that it is. Unless a question specifically deals with compiler #include directives, you should assume the necessary header files have been included.
- Be careful to distinguish integer values from floating point values (containing a decimal point). In questions/answers that require a distinction between integer and real values, integers will be represented without a decimal point, whereas real values will have a decimal point, [1704 (integer), 1704.0 (floating point)].
- **This is a closed-book, closed-notes examination.**
- **No laptops, calculators or other electronic devices may be used during this examination.**
- **You may not discuss (in any form: written, verbal or electronic) the content of this examination with any student who has not taken it.**
- **You must return this test form when you complete the examination. Failure to adhere to any of these restrictions is an Honor Code violation.**
- There are 20 multiple-choice questions and one design/implementation question, priced as marked.
- The answers you mark on the Opscan form will be considered your official answers.
- When you have finished, sign the pledge at the bottom of this page and turn in the test and your Opscan.

**Do not start the test until instructed to do so!**

**Name (Last, First)** _____

printed

**Pledge:** On my honor, I have neither given nor received unauthorized aid on this examination.

_____

*signature*

1. [4 pts] Which of these are syntactic purposes of having a header file associated with (not #included by) a particular source file?

   1) To "import" declarations of names that are declared elsewhere into a scope in which those names are to be used.
   2) To "import" declarations of all of the names that to be used into a scope.
   3) To "export" the declarations of those entities defined in the source file that need to be used in another compilation unit.
   4) To "export" the declarations of all of the entities defined in the source file.
   5) To justify the inclusion of `#ifndef` and `#endif` in the C++ language.

   6) 1 and 3 only
   7) 2 and 4 only
   8) 1 and 4 only
   9) 2 and 3 only
   10) None of these

---

2. [4 pts] Which of the following are syntactic purposes of the `#include` mechanism in C++?

   1) To "import" declarations of names that are declared elsewhere into a scope in which those names are to be used.
   2) To "import" declarations of all of the names that to be used into a scope.
   3) To "export" the declarations of those entities defined in the source file that need to be used in another compilation unit.
   4) To "export" the declarations of all of the entities defined in the source file.
   5) To justify the inclusion of `#ifndef` and `#endif` in the C++ language.

   6) 1 and 3 only
   7) 2 and 4 only
   8) 1 and 4 only
   9) 2 and 3 only
   10) None of these

---

3. [4 pts] Suppose that a class `Node` is implemented by placing the class declaration in a file `Node.h` and the implementations of the class member functions in another file `Node.cpp`. In order to be able to compile the file `Node.cpp`:

   1) `Node.cpp` must contain an include directive for `Node.h`, because `Node` will be used as a type name in `Node.cpp`.
   2) `Node.cpp` must contain an include directive for `Node.h`, but for some other reason than given in 1.
   3) `Node.cpp` must not contain an include directive for `Node.h`.
   4) `Node.cpp` may or may not contain an include directive for `Node.h`; it doesn't matter.
   5) None of these

---

4. [4 pts] Effects of intelligent use of separate compilation include:

   1) reduced compilation/link time for large projects after implementation changes in one function or class.
   2) increased compilation/link time for large projects after implementation changes in one function or class.
   3) easier re-use of independent code modules, such as data structures and data types.
   4) harder re-use of independent code modules, such as data structures and data types.
   5) 1 and 3 only
   6) 2 and 4 only
   7) None of these

---

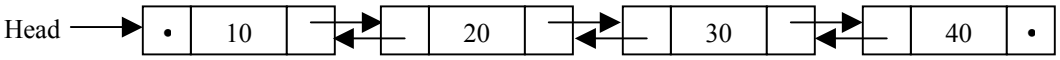
5. [4 pts] Failing to use conditional compilation directives correctly can result in:

   1) Errors due to declarations or definitions of names not being in scope.
   2) Errors due to multiple occurrences of declarations or definitions of the same name within a single scope.
   3) Both of these
   4) None of these

For questions 6 and 7, assume the doubly-linked node class shown below, and that the list structure shown below has been created. (`Head` is of type `Node*`.)

```
class Node {
public:
   int   Element;
   Node *Prev;      // points toward head of list
   Node *Next;      // points away from head of list

   Node(int E = 0, Node* P = NULL, Node* N = NULL);
   ~Node();
};
```




6. [6 pts] Which, if any, of the code fragments given below would delete the second node from the list structure given above, and leave the remainder of the list correctly connected (including `Head`)?

1)
```
Node* Tmp = Head->Next;
delete Tmp;
Head->Next = Head->Next->Next;
Head->Next->Next->Prev = Head;
```

2)
```
Node* Tmp = Head->Next;
Head->Next = Head->Next->Next;
Head->Next->Next->Prev = Head;
delete Tmp;
```

3)
```
Node* Tmp = Head->Next;
Head->Next = Tmp->Next;
Tmp->Next->Prev = Tmp->Prev;
delete Tmp;
```

4) All of them

5) 1 and 2 only

6) 1 and 3 only

7) 2 and 3 only

8) None of these

7. [6 pts] Which, if any, of the code fragments given below would insert a new node containing the value 25 between the second and third nodes in the original list structure given above?

1)
```
Node* Tmp = new Node(25);
Tmp->Prev = Head->Next;
Tmp->Next = Head->Next->Next;
Head->Next->Next->Prev = Tmp;
Head->Next->Next = Tmp;
```

2)
```
Node* Tmp = new Node(25);
Head->Next->Next->Prev = Tmp;
Head->Next->Next = Tmp;
Tmp->Prev = Head->Next;
Tmp->Next = Head->Next->Next;
```

3)
```
Node* Tmp = Head;
while ( Tmp->Next != NULL )
   Tmp = Tmp->Next;
Tmp->Next = new Node(Tmp.Element);
Tmp->Next->Prev = Tmp;
Tmp->Element = Tmp->Prev->Element;
Tmp->Prev->Element = 25;
```

4) All of them

5) 1 and 2 only

6) 1 and 3only

7) 2 and 3 only

8) None of these

For questions 8 through 10, consider the following list class designed to store integers:

```
class SList {
private:
   SNode* Head;
   SNode* Tail;
   SNode* Current;

public:
   SList();                                 // make an empty list
   SList(const SList& Source);              // copy constructor
   SList& operator=(const SList& RHS); // assignment overload
   bool  Insert(const int& E);              // insert value E at current position
   bool  Delete(int & E);                   // delete value at current position
   int&  Get() const;                       // get reference to current data element

   bool  Advance();                         // move current position toward tail
   void  goToHead();                        // move current position to head
   void  goToTail();                        // move current position to tail
   bool  atEnd() const;                     // true if current position is NULL

   bool  isEmpty() const;                   // true if list is empty
   void  Display(ostream& Out) const;  // print list contents

   ~SList();                                // deallocate nodes
};
```

A client function is implemented to take replace each element stored in an `SList` object with its absolute value.

```
void abs(SList& L) {                    // Line 1

   ____________________                 // Line 2

   while ( !L.atEnd() ) {               // Line 3

      ________________________          // Line 4
      ________________________          // Line 5

      L.Advance();                      // Line 6
   }
}
```

8. [4 pts] How should the blank in Line 2 be filled to prepare the list for processing?

   1) It should be left blank.
   2) `L.goToTail();`
   3) `L.goToHead();`
   4) `L.Advance();`
   5) None of these

9. [4 pts] How should the blanks in Lines 4 and 5 be filled to modify the elements stored in the list as specified?

   1) They should be left blank.

   2) `L.Get() = abs(L.Get());`
      (Line 5 left blank)

   3) `int& Tmp = L.Get();`
      `Tmp = abs(Tmp);`

   4) `int Tmp = L.Get();`
      `Tmp = abs(Tmp);`

   5) All of them
   6) 2 and 3 only
   7) 2 and 4 only
   8) 3 and 4 only
   9) None of these

10. [4 pts] How should the body of the destructor for `SList` be filled?

1)
```
while ( Head != NULL ) {
   delete Head;
   Head = Head->Next;
}
```

2)
```
SNode* Tmp = Head;
while ( Tmp != Tail ) {
   Head = Head->Next;
   delete Tmp;
   Tmp = Head;
}
```

3)
```
SNode* Tmp = Head;
while ( Tmp != NULL ) {
   Head = Head->Next;
   delete Tmp;
   Tmp = Head;
}
```

4)
```
delete Head;
```

5) `SList` doesn't need a destructor.

6) 1, 2 and 3 only

7) 1 and 2 only

8) 1 and 3 only

9) 2 and 3 only

10) None of these

---

For questions 11 through 13, determine which of the following big-O categories the given function belongs to:

1) `1`
2) `log(n)`
3) `n`
4) `n log(n)`
5) `n^2`
6) `2^n`
7) None of these

11. [4 pts] `3n^2 + 5n log(n) + 1000`

12. [4 pts] `1000 + 5 log(n)`

13. [4 pts] `3n + 5n log(n) + 1000`

---

14. [6 pts] Suppose there are two algorithms for solving a problem, the first with complexity `O(n^2)` and the second with complexity `O(n log n)`. If the input size is `2^16`, estimate how much longer should the first algorithm would take than the second algorithm?

1) The same amount of time.
2) About twice as long.
3) About 10 times as long.
4) About 50 times as long.
5) About 100 times as long.
6) About 500 times as long.
7) About 1000 times as long.
8) About 5000 times as long.
9) About 10,000 times as long.
10) None of these.

---

15. [4 pts] Choosing your answer from those given above for questions 11 through 13, what is the complexity of the following code fragment?

```
for (int j = 0; j < n; j++) {
   for (int k = 0; k < 1000; k++) {
      cout << n;
   }
   cout << endl;
}
```

16. [4 pts] Consider calling the recursive function below with the parameter value 6. What value would be returned?

```
int Annoy(int K) {
   if ( K == 1 )
      return 1;
   if ( K % 2 != 0 )
      return ( 1 + Annoy( 3 * K + 1) );
   else
      return ( 2 + Annoy( K / 2 ) );
}
```

1) 1
2) 2
3) 3
4) 8
5) 11
6) 12
7) 15
8) 16
9) None, infinite recursion.
10) None of these.

---

For questions 17 and 18, consider the following function which is intended to return the number of occurrences of the value `Target` in the array `A`:

```
int Count(int Target, const int A[], int Start, int Stop) {   // Line 1

   if ( Start < 0 || Stop < 0 || Start > Stop ) _________      // Line 2

   if ( A[Start] == Target )
      ____________________________________________________     // Line 3
   else
      ____________________________________________________     // Line 4
}
```

17. [4 pts] How should the blank in Line 2 be filled?

1) It should be left blank.
2) `return;`
3) `return 0;`
4) `Start = 0;`
5) `Start++;`
6) `Stop--;`
7) `return Start;`
8) `return Stop;`
9) None of these.

18. [4 pts] How should the blank in Line 3 be filled?

1) It should be left blank.
2) `return 0;`
3) `return 1;`
4) `Count(Target, A, Start + 1, Stop);`
5) `return (Count(Target, A, Start + 1, Stop) );`
6) `1 + Count(Target, A, Start + 1, Stop);`
7) `return ( 1 + Count(Target, A, Start + 1, Stop) );`
8) None of these.

---

19. [4 pts] Comparing an iterative and a recursive implementation of an algorithm (and ignoring compiler issues):

1) The iterative version is typically faster.
2) The recursive version is typically faster.
3) The iterative version is typically shorter.
4) The recursive version is typically shorter.
5) 1 and 3 only
6) 1 and 4 only
7) 2 and 3 only
8) 2 and 4 only
9) None of these.

20. [4 pts] Consider modifying the `SList` class given earlier to include a public function with declaration:

```
int Size() const;
```

that returns the number of nodes in the list. Just for fun, suppose that the implementation is to use a recursive design. Will adding this feature also require adding a "helper" function for `Size()`?

1) No, `Size()` must be entirely self-contained and there is no role for a "helper" function.
2) No, although using one would produce a more elegant result.
3) No, in fact this cannot be done at all because class member functions are not allowed to be recursive.
4) Yes, because the recursion requires passing information that the client could not supply when calling a member function.
5) None of these.

---

21. [14 pts] Assume a queue class, implemented to store values of type `int`, with the following interface:

```
class Queue {
private:
   // irrelevant

public:
   Queue();                                 // make empty queue
   Queue(const Queue& Source);              // copy queue
   Queue& operator=(const Queue& RHS);      // assign queue
   int  deQueue();                          // delete elem at front
   void enQueue(int Value);                 // insert elem at rear
   int& Peek();                             // access elem at front
   bool Empty() const;                      // is queue empty?
   bool Full() const;                       // is queue full
   void Clear();                            // delete queue contents
   ~Queue();                                // destroy queue
};
```

While it is not the most natural approach, a set of integers can be represented by storing its elements in a `Queue` object. Given two `Queue` objects, `A` and `B`, their intersection is the queue that contains all of the values that occur both in `A` and in `B`, and no other elements. For example, the intersection of the queues {8, 14, 7, 18, 6} and {7, 19, 3, 8} is {7, 8}. The order in which the elements are stored does not matter, but duplicate entries are not allowed. You may assume that the two given queues are free of duplicates.

Your problem is to design and implement a function that will take two queues and compute their intersection. The function must conform to the following prototype (note it's NOT a member function):

```
Queue Intersection(Queue A, Queue B);
```

Obviously, the most important criterion is that your function computes the correct result. Efficiency also counts; you should take advantage of the relevant interface features in the `Queue` class, rather than re-implementing them unnecessarily.

Minor syntax errors will not be penalized as long as the intent is clear and correct. You should include comments explaining the significance of the major blocks of code in your implementation. Neatness does count; if your answer is so illegible or so poorly organized that it is difficult to understand, you will lose points for that. Possibly many points.

You may write one or more helper functions if you find that useful, but it is not required.

```
Queue Intersection(Queue A, Queue B) {
```