Virginia Tech

1872

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 preprocessor #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 22 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 source (cpp) file #include a particular header file?

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

5) All of these
6) 1 and 2 only
7) 1 and 3 only
8) 2 and 3 only
9) None of these

2. [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

3. [4 pts] Which of the following are not effects of intelligent use of separate compilation?

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

4. [4 pts] Conditional compilation directives, when used 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) Increased compilation times because the compiler must process extra code.
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

5. [4 pts] In the following code fragment, at which line does the pointer Q first have a target?

```
string* Q;       // Line 1
Q = NULL;        // Line 2
Q = new string;  // Line 3
```

1) Line 1
2) Line 2
3) Line 3
4) Q never has a target.
5) None of these

For questions 6 through 10, consider the following code fragment:

```
int* p = NULL;        // Line 1
int* q = NULL;        // Line 2
```

```
   p = new int;         // Line 3
   *p = 17;             // Line 4
   q = p;               // Line 5
   *q = 42;             // Line 6
   delete q;            // Line 7
```

6) What is the effect of the statement in Line 4?

1) To create a target for the pointer p.
2) To assign the pointer p a new value.
3) To assign the target of the pointer p a value.
4) All of these.
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

7) What is the effect of the statement in Line 5?

1) To create a target for the pointer p.
2) To assign the pointer p a new value.
3) To assign the target of the pointer p a value.
4) All of these.
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these

8) After Line 5 is executed, what is the value of the target of q?

1) q doesn't have a target.
2) q has a target but it's uninitialized.
3) 17
4) 42
5) 2 or 3 only
6) 2 or 4 only
7) 3 or 4 only
8) None of these

9) After Line 6 is executed, what is the value of the target of p?

1) p doesn't have a target.
2) p has a target but it's uninitialized.
3) 17
4) 42
5) 2 or 3 only
6) 2 or 4 only
7) 3 or 4 only
8) None of these

10) After Line 7 is executed, what is the value of the target of p?

1) p has a target but it's uninitialized.
2) 17
3) 42
4) p no longer has a target.
5) 1 or 2 only
6) 1 or 3 only
7) 2 or 3 only
8) None of these

For questions 11 and 12, consider the following function, which is intended to allocate a new array of integers of the specified size, and initialize each cell of the array to the specified value, and make it available to the caller.

```
// Preconditions:
//    - Sz has been initialized to the desired dimension
//    - Value has been initialized to the desired preset value
// Postconditions:
//    - List points to an array of Sz ints, each set to equal Value
//
void makeArray(________ List, unsigned int Sz, int Value) {  // Line 1

   List = __________________;                          // Line 2:  create array
   for (unsigned int Pos = 0; Pos < Sz; Pos++)         // Line 3:  init array
      List[Pos] = Value;                               // Line 4
}
```

11. [4 pts] How should the type for the first parameter in Line 1 be specified?

1) `int`
2) `int*`
3) `int&`
4) `int*&`
5) 2 or 3 only
6) 3 or 4 only
7) None of these

12. [4 pts] How should the value assigned to `List` in Line 2 be computed?

1) `new int`
2) `new int[Sz]`
3) `int[Sz]`
4) 1 or 2 only
5) 1 or 3 only
6) 2 or 3 only
7) None of these

For questions 13 and 14, consider the following function, which is intended to safely deallocate an array which has been created by calling the function `makeArray()` given above:

```
void destroyArray(int*& List) {   // Line 1

   __________________;            // Line 2: deallocate the array
   __________________;            // Line 3: eliminate the dangling pointer
}
```

13. [4 pts] What statement should be used in Line 2?

1) `List = NULL`
2) `delete [] List`
3) `delete List`
4) Any of them would do.
5) 1 or 2 only
6) 1 or 3 only
7) 2 or 3 only
8) None of these

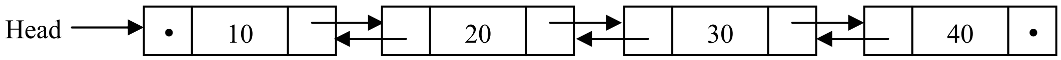
14. [4 pts] What statement should be used in Line 3?

1) `List = NULL`
2) `delete [] List`
3) `delete List`
4) Any of them would do.
5) 1 or 2 only
6) 1 or 3 only
7) 2 or 3 only
8) None of these

For questions 15 and 16, 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();
};
```



15. [4 pts] Which, if any, of the code fragments given below would properly delete the first node from the list structure given above, and leave the remainder of the list correctly connected (including `Head`)?

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

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

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

16. [4 pts] Which, if any, of the code fragments given below would insert a new node containing the value 25 <u>in front of</u> the first node in the <u>original</u> list structure given above?

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

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

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

4) All of them

5) 1 and 2 only
6) 1 and 3only
7) 2 and 3 only
8) None of these

For questions 17 through 22, 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
};
```

SList uses a class SNode which is identical to the Node class shown earlier except that it only has one pointer. A member function is implemented to perform a circular shift on an SList; that is, each element moves forward one spot (toward the head) and the element at the head moves to the tail:

```
   void SList::Circulate() {
      if ( Head == NULL ) return;  // 1: Nothing to shift if ( Head == Tail )
   return;  // 2: No need to shift SNode* Save;
      ____________________;        // 3: Don't lose first node
      ____________________;        // 4: Move 2nd node to front
      Save->Next = NULL;           // 5: Update old first node
      ____________________;        // 6: Move old first node to end
      ____________________;        // 7: Update tail
   }
```

Note: it is possible to complete the code above so that it will work correctly, without adding any additional statements other than the ones represented by blanks.

17. [4 pts] How should the blank in Line 3 be filled?

1) It should be left blank.
2) `Head = Save`
3) `Save = Head`
4) `Save = new Node`
5) None of these

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

1) `Head = Save`
2) `Head = Head->Next`
3) `Head = NULL`
4) `Head = Save->Next`
5) All of them would do.
6) Only 2 or 4 would do.
7) None of these

19. [4 pts] How should the blank in Line 6 be filled?

    1) `Tail = Save`
    2) `Save->Next = Tail`
    3) `Tail = Save->Next`
    4) All of them would do.
    5) It should be left blank.
    6) Only 1 or 2 would do.
    7) Only 1 or 3 would do.
    8) None of these

20. [4 pts] How should the blank in Line 7 be filled?

    1) `Tail = Save`
    2) `Save->Next = Tail`
    3) `Tail = NULL`
    4) All of them would do.
    5) It should be left blank.
    6) Only 1 or 2 would do.
    7) Only 1 or 3 would do.
    8) None of these

21. [4 pts] According to the declaration of `SList`, the implementation of `operator=` returns something of type `SList&`. What does this accomplish?

    1) Nothing, `operator=` doesn't need to return a value.
    2) It prevents errors if an `SList` object is assigned to itself.
    3) It allows "chaining" of assignment operations together.
    4) It completes the copying of the parameter into the target object.
    5) None of these

22. [4 pts] According to the declaration of `SList`, the parameter to the copy constructor is passed by constant reference. What is the best reason it is not passed by value?

    1) No reason at all; it might as well be passed by value.
    2) Passing it by value would require extra time and memory.
    3) Passing by value would require less time and/or memory, so it should be done that way.
    4) The copy constructor defines how an `SList` object is passed by value.
    5) None of these

---

23. [12 pts] Write the implementation of the assignment operator overload for the `SList` class declared earlier.

    Be sure to take into account the discussion of the logic of deep copy in class.
    Explain the purpose of each statement with a clear, brief comment.
    The syntax of your answer will not be graded strictly, but the syntax must be close enough to be comprehensible.

Write your answer on the following page!

```
SList& SList::operator=(const SList& RHS) {

   // test for self-assignment
   if ( this == &RHS ) return (*this);

   // delete the nodes from the target list
   Current = Head;
   while ( Head != NULL ) {
      Head = Head->Next;
      delete Current;
      Current = Head;
   }
   Head = Tail = Current = NULL;

   // duplicate the contents of RHS
   SNode* toCopy = RHS.Head;          // get a handle on source list

   while ( toCopy != NULL ) {
      SNode* Copy = new SNode(toCopy->Element);  // duplicate node

      if ( Head == NULL ) {           // put node at end of list
         Head = Tail = Copy;
      }
      else {
         Tail->Next = Copy;
         Tail = Copy;
      }

      if ( toCopy == RHS.Current )    // preserve current position
         Current = Copy;

      toCopy = toCopy->Next;          // step to next node in source
   }

   return ( *this );                  // return the copy for chaining

}
```