Capitulo 11

This activity contains 29 questions.

1. *Section 11.1 Introduction*

11.1 Q1: Which of the following is not an operator overloaded by the C++ language?

- ○ >>.
- ○ pow.
- ○ <<.
- ○ +.

2. *Section 11.2 Fundamentals of Operator Overloading*

11.2 Q1: To use an operator on user-defined class objects, operator overloading:

- ○ Must always be used.
- ○ Must always be used, with three exceptions.
- ○ Must never be used.
- ○ Must never be used, with three exceptions.

3. *11.2 Q2: The correct function name for overloading the addition (+) operator is:*

- ○ operator+.
- ○ operator(+).
- ○ operator_+.
- ○ operator:+.

4. *Section 11.3 Restrictions on Operator Overloading*

11.3 Q1: Which of the following operators cannot be overloaded?

- ○ The -> operator.
- ○ The . operator.
- ○ The & operator.
- ○ The [ ] operator.

5. *11.3 Q2: Which statement about operator overloading is false?*

- ○ *The precedence of an operator cannot be changed by overloading.*
- ○ *Certain overloaded operators can change the number of arguments they take.*
- ○ *New operators can never be created.*
- ○ *Overloading cannot change how an operator works on built-in types.*

6. *11.3 Q3: To implicitly overload the += operator:*

- ○ *The += operator cannot be overloaded implicitly.*
- ○ *Both the + and = operators need to be overloaded.*
- ○ *Only the + operator needs to be overloaded.*
- ○ *Only the = operator needs to be overloaded.*

7. *Section 11.4 Operator Functions as Class Members vs. Global Functions*

11.4 Q1: Which of the following operators can be overloaded as a global function?

- ○ *==.*
- ○ *().*
- ○ *[].*
- ○ *+=.*

8. *11.4 Q2: Which situation would require the operator to be overloaded as a global function?*

- ○ *The overloaded operator is =.*
- ○ *The left most operand must be a class object (or a reference to a class object).*
- ○ *The left operand is an int.*
- ○ *The operator returns a reference.*

9. *11.4 Q3: An overloaded + operator takes a class object and a double as operands. For it to be commutative (i.e., a + b and b + a both work):*

- ○ *operator+ must be a member function of the class from which the objects are* *instantiated.*

- ○ *It must be overloaded twice; the operator+ function that takes the object as the left operand must be a member function, and the other operator+ function must be a global function.*
- ○ *The + operator cannot be overloaded to be commutative.*
- ○ *operator+ must be a non-member function.*

10.

Section 11.5 Overloading Stream Insertion and Stream Extraction Operators

11.5 Q1: Suppose you have a programmer-defined data type Data and want to overload the << operator to output your data type to the screen in the form cout << dataToPrint; and allow cascaded function calls. The first line of the function definition would be:

- ○ *ostream &operator<<(const Data &dataToPrint, ostream &output).*
- ○ *ostream operator<<(const Data &dataToPrint, ostream &output).*
- ○ *ostream &operator<<(ostream &output, const Data &dataToPrint).*
- ○ *ostream operator<<(ostream &output, const Data &dataToPrint).*

11.

Section 11.6 Overloading Unary Operators

11.6 Q1: Suppose the unary ! operator is an overloaded member function of class String. For a String object s, which function call is generated by the compiler when it finds the expression !s?

- ○ *s.operator!().*
- ○ *s.operator!(default_value1, default_value2,…).*
- ○ *operator!(s).*
- ○ *A compiler error results because no arguments are given.*

12.

Section 11.7 Overloading Binary Operators

11.7 Q1: y and z are user-defined objects and the += operator is an overloaded member function. The operator is overloaded such that y += z adds z and y, then stores the result in y. Which of the following expressions is always equivalent to y += z?

- ○ *y.operator+=(z).*
- ○ *y = y operator+= z.*
- ○ *y operator+=(y + z).*
- ○ *y = y + z.*

13. *11.7 Q2: For operators overloaded as non-static member functions:*

- ○ *Both binary and unary operators take one argument.*
- ○ *Binary operators can have two arguments and unary operators can have one.*
- ○ *Neither binary nor unary operators can have arguments.*
- ○ *Binary operators can have one argument, and unary operators cannot have any.*

14. *Section 11.8 Case Study: Array Class*

11.8 Q1: Which of the following is false?

- ○ *Two arrays cannot be meaningfully compared with equality or relational operators.*
- ○ *C++ ensures that you cannot "walk off" either end of an array.*
- ○ *Arrays cannot be assigned to one another (i.e., array1 = array2;).*
- ○ *An entire non-char array cannot be input or output at once.*

15. *11.8 Q2: The array subscript operator [], when overloaded, cannot:*

- ○ *Take a float as an operand.*
- ○ *Take multiple values inside (e.g., [4 8]).*
- ○ *Be used with linked list classes.*
- ○ *Take user-defined objects as operands.*

16. *11.8 Q3: A copy constructor:*

- ○ *None of the above.*
- ○ *Is a constructor with only default arguments.*
- ○ *Is a constructor that takes no arguments.*
- ○ *Is a constructor that initializes a newly declared object to the value of an existing object of the same class.*

17. *11.8 Q4: Copy constructors must receive its argument by reference because:*

- ○ *Otherwise the constructor will only make a copy of a pointer to an object.*
- ○ *The copy of the argument passed by value has function scope.*
- ○ *Otherwise infinite recursion occurs.*
- ○ *The pointer needs to know the address of the original data, not a temporary copy of it.*

18. *11.8 Q5: To prevent class objects from being copied:*

- ○ *Make the overloaded assignment operator private.*
- ○ *None of the above.*
- ○ *Make the copy constructor private.*
- ○ *Both (a) and (b).*

19. *Section 11.9 Converting between Types*

11.9 Q1: Conversion constructors:

- ○ *Can convert between user-defined types.*
- ○ *Cannot convert built-in types to user defined types.*
- ○ *Are implicitly defined by the compiler if not explicitly written by the programmer.*
- ○ *Can have multiple arguments.*

20. *11.9 Q2: The prototypes of overloaded cast operator functions do not:*

- ○ *Specify a return type.*
- ○ *Specify the type they convert to.*
- ○ *Need to be defined inside the class whose objects are being converted.*
- ○ *Specify the type that is being converted.*

21. *11.9 Q3: Which of the following lines would be the prototype for an overloaded cast operator function that converts an object of user-defined type Time into a double?*

- ○ *Time::operator double() const;.*
- ○ *Time::operator_cast(double) const;.*
- ○ *Time::static_cast double() const;.*
- *d. Time::double() const;.*

○

22.

Section 11.10 Case Study: String Class

11.10 Q1: Conversion constructors cannot:

○ *Be applied implicitly.*

○ *Be used to convert the arguments for overloaded operators to the types needed by those overloaded operators.*

○ *Be used implicitly in series to match the needs of an overloaded operator.*

○ *Take exactly one argument.*

23.

11.10 Q2: Which of the following is not a disadvantage of default memberwise copy with objects containing pointers?

○ *Having the possibility of leaving a dangling pointer.*

○ *Requiring the explicit overloading of the assignment operator.*

○ *Allowing the destructor of one object to be called while leaving the second pointer, to the same memory location, intact.*

○ *Allowing both objects to point to the same dynamically allocated storage.*

24.

11.10 Q3: Assume that the function call operator() is overloaded for data type String in the usual sense of selecting a substring from a larger string. For a String object string1 with the character string "ABCDEFGHI", what string does string1(4 , 2) return?

○ *"EFGHI".*

○ *"CDEF".*

○ *"EF".*

○ *"CD".*

25.

Section 11.11 Overloading ++ and --

11.11 Q1: The conventional way to distinguish between the overloaded preincrement and postincrement operators (++) is:

○ *To make the argument list of postincrement include an int.*

○ *To assign a dummy value to preincrement.*

○ *To have the postincrement operator call the preincrement operator.*

- ◯ *Implicitly done by the compiler.*

26. *11.11 Q2: Because the postfix increment operator returns objects by value and the prefix increment operator returns objects by reference:*

- ◯ *Objects returned by postfix increment cannot be used in larger expressions.*
- ◯ *The postfix increment operator typically returns a temporary object that contains the original value of the object before the increment occurred.*
- ◯ *The postfix increment operator returns the actual incremented object with its new value.*
- ◯ *Prefix increment has slightly more overhead than postfix increment.*

27. *Section 11.12 Case Study: A Date Class*

11.12 Q1: There exists a data type Date with member function Increment that increments the current Date object by one. The ++ operator is being overloaded to postincrement an object of type Date. Select the correct implementation:

◯
```
Date Date::operator++( int )
{
   Date temp = *this;
   Increment();
   return *temp;
}.
```

◯
```
Date Date::operator++( int )
{
   Date temp = *this;
   return this;
   temp.Increment();
}.
```

◯
```
Date Date::operator++( int )
{
   Date temp = *this;
   Increment();
   return temp;
}.
```

◯
```
Date Date::operator++( int )
{
   Increment();
   Date temp = *this;
```

```
      return temp;
   }.
```

28. *Section 11.13 Standard Library Class string*

11.13 Q1: Which of the following is false?

- ◯ *A string can be defined to store any data type.*
- ◯ *d. An exception is thrown. if the argument to string's at member function is an invalid subscript.*
- ◯ *Class string's overloaded [] operator returns a vector element as an rvalue or an lvalue, depending on the context.*
- ◯ *b. Class string provides bounds checking in its member function at.*

29. *Section 11.14 explicit Constructors*

11.14 Q1: An explicit constructor:

- ◯ *Cannot be called outside of the class it is declared in.*
- ◯ *Can be implicitly called by the compiler to perform a data type conversion.*
- ◯ *Does not initialize its class's data members.*
- ◯ *Must take exactly one argument.*

Clear Answers / Start Over   Submit Answers for Grading

Answer choices in this exercise appear in a different order each time the page is loaded.