Total Score:

COP 3331-01: Object-Oriented Design

Instructor: Dr. Henrick Jeanty Exam #1, November 14, 2012

°Welcome to exam #1 in Object-Oriented Design. You have 75 minutes. Read each problem carefully. You may have with you (on your desk, that is) pencils, erasers, and a lucky rabbit’s foot. Please answer on the sheets and on the back if you need to. Submit everything in problem order. Good luck Note that although the test is graded out of 120 points, there are a total of 128 points for an extra 8 points.

|  |  |
| --- | --- |
| Name: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

1: \_\_\_\_\_\_\_\_ models software in terms similar to those that people use to describe real-world objects.

1. Object-oriented programming
2. Object-oriented design
3. Procedural programming
4. None of the above

ANS: b. Object-oriented design.

2: Which statement is *false*?

a. Classes are reusable software components.

b. A class is to an object as a blueprint is to a house.

c. Performing a task in a program requires a method.

d. A class is an instance of its object.

ANS: A class is an instance of its object. The reverse is true.

3: The creates object code and stores it on disk.

1. Interpreter.
2. Compiler.
3. Preprocessor.
4. Loader.

**ANS: b. Compiler.**

4: The linker links:

1. The source code with the object code.
2. The object code with the external libraries.
3. The executable code with primary memory.
4. The primary memory with the CPU.

**ANS: b. The object code with external libraries.**

5: Which of the following statements would display the phrase C++ is fun?

1. std::cout << "Thisis fun\rC++ ";
2. std::cout << '++ is fun';
3. std::cout << "\"C++ is fun\"";
4. std::cout << C++ is fun;

**ANS: a.** std::cout << "Thisis fun\rC++ ";

6: Which of the following is a variable declaration statement?

1. int total;
2. #include <iostream>
3. int main()
4. // first string entered by user

**ANS: a. int total;**

7: The \_\_\_\_\_\_\_\_ object enables a program to read data from the user.

1. std::cout.
2. std::cin.
3. std::cread.
4. std::cget.

**ANS:b. std::cin.**

8: What will be the output after the following C++ statements have been executed?

int a, b, c, d;

a = 4;

b = 12;

c = 37;

d = 51;

if ( a < b )

cout << "a < b" << endl;

if ( a > b )

cout << "a > b" << endl;

if ( d <= c )

cout << "d <= c" << endl;

if ( c != d )

cout << "c != d" << endl;

1. a < b  
   c != d
2. a < b  
   d <= c   
   c != d
3. a > b   
   c != d
4. a < b  
   c < d   
   a != b

**ANS: a.** a < b

c != d

9: Which of the following *is* a compilation error?

1. Neglecting to declare a local variable in a function before it is used.
2. Using a triple equals sign instead of a double equals sign in the condition of an if statement.
3. Omitting the left and right parentheses for the condition of an if statement.
4. All of the above.

**ANS: d. All of the above.**

**Section 3.2 Defining a Class with a Member Function**

10: C++ functions other than main are executed:

1. Before main executes.
2. After main completes execution.
3. When they are explicitly called by another function.
4. Never.

**ANS c. When they are explicitly called by another function.**

11: Function headers contain *all* of the following *except*:

1. Parentheses.
2. Left brace.
3. Name of function.
4. Return type.

**ANS: b. Left brace.**

12: Calling a member function of an object requires which item?

1. The dot operator.
2. Open and close braces.
3. The class name.
4. None of the above.

**ANS a. The dot operator.**

**Section 3.4 Data Members, *set* Functions and *get* Functions**

13: Attributes of a class are also known as:

1. Constructors.
2. Local variables.
3. Data members.
4. Classes.

**ANS: c. Data members.**

14: What is the default initial value of a String?

1. ""
2. "default"
3. default
4. None of the above.

**ANS: a.** ""

15: What type of member functions allow a client of a class to assign values to private data members?

1. *Client* member functions.
2. *Get* member functions.
3. *Set* member functions.
4. None of the above.

**ANS: c. *Set* member functions.**

**Section 3.5 Initializing Objects with Constructors**

16: A *default* constructor has how many parameters?

1. 0.
2. 1.
3. 2.
4. Variable number.

**ANS: a. 0.**

17: A constructor can specify the return type:

1. int.
2. string.
3. void.
4. A constructor cannot specify a return type.

**ANS: d. A constructor cannot specify a return type.**

18: The compiler will *implicitly* create a default constructor if:

1. The class does not contain any data members.
2. The programmer specifically requests that the compiler do so.
3. The class does not define any constructors.
4. The class already defines a default constructor.

**ANS: c. The class does not define any constructors.**

**Section 3.6 Placing a Class in a Separate File for Reusability**

19: A header file is typically given the filename extension:

1. .h
2. .hdr
3. .header
4. .cpp

**ANS: a.** .h

20: When a client code programmer uses a class whose implementation is in a separate file from its interface, that implementation code is merged with the client’s code during the:

1. Programming phase.
2. Compiling phase.
3. Linking phase.
4. Executing phase.

**ANS: c.** Linking phase**.**

**Section 4.8 Formulating Algorithms: Counter-Controlled Repetition**

21: An uninitialized local variable contains:

1. The value last stored in the memory location reserved for that variable.
2. No value.
3. A value of zero.
4. A randomly assigned value.

**ANS: a. The value last stored in the memory location reserved for that variable.**

**Section 5.2 Essentials of Counter-Controlled Repetition**

22: Which of the following does counter-controlled repetition *require*?

1. An initial value.
2. A condition that tests for the final value.
3. An increment or decrement by which the control variable is modified each time through the loop.
4. All of the above.

**ANS d. All of the above.**

**Section 5.3** for **Repetition Statement**

23: If a variable is declared in the initialization expression of a for statement, then:

1. It is automatically reinitialized to zero once the loop is finished.
2. The scope of the variable is restricted to that for loop.
3. It retains its final value after the loop is finished.
4. It can not be used in any structures that are nested in that forstructure.

**ANS b. The scope of the variable is restricted to that particular** for **loop.**

24: Which of the following is *false*?

1. The three expressions in the for structure are optional.
2. The initialization and increment expressions can be comma-separated lists.
3. You must declare the control variable outside of the for loop.
4. A for loop can always be used to replace a while loop, and vice versa.

**ANS c. You must declare the control variable outside of the** for **loop.**

**Section 5.4 Examples Using the** for **Statement**

25: Which of the following for headers is *not* valid?

1. for ( int i = 0; i < 10; i++ )
2. int i = 0;  
   for ( ; i < 10; i++ )
3. for ( int i = 0; int j = 5; ; i++ )
4. All of the above.

**ANS c.** for ( int i = 0; int j = 5; ; i++ )**.**

**Section 6.2 Program Components in C++**

26: All of the following are *true* of functions except:

1. They define specific tasks that can be used at many points in a program.
2. A function call must specify the name and arguments of the function.
3. The definition of a function usually is visible to other functions.
4. The implementation of a function is hidden from the caller.

**ANS c. The definition of a function usually is visible to other functions.**

27: Functions can:

1. Be used as building blocks to create new programs.
2. Return a result to the caller function.
3. Be reused any number of times.
4. Do all of the above.

**ANS d. Do all of the above.**

28: Using the following function definition, the parameter list is represented by:

A B ( C )

{

D

}

1. A.
2. B.
3. C.
4. D.

**ANS c.** C**.**

29: A function prototype does *not* have to:

1. Include parameter names.
2. Terminate with a semicolon.
3. Agree with the function definition.
4. Match with all calls to the function.

**ANS: a. Include parameter names.**

30: A function prototype can always be omitted when:

1. A function is defined before it is first invoked.
2. A function is invoked before it is first defined.
3. A function takes no arguments.
4. A function does not return a value.

**ANS: a. A function is defined before it is first invoked.**

31: Converting from type \_\_\_\_\_\_\_\_ to type \_\_\_\_\_\_\_\_ will result in the loss of data.

1. bool, char.
2. float, double.
3. int, char.
4. short, long.

**ANS: c.** int**,** char**.**

**Section 6.7 Case Study: Random Number Generation**

32: The randfunction generates a data value of the type:

1. unsigned int.
2. int.
3. long int.
4. short int.

**ANS: a.** unsigned int.

33: A variable that can have values *only* in the range 0 to 65535 is a:

1. Four-byte int.
2. Four-byte unsigned int.
3. Two-byte int.
4. Two-byte unsigned int.

**ANS: d. Two-byte** unsigned int.

34: In the expression

n = x + rand() % y;

1. y is the shifting value.
2. x is the scaling value.
3. y is the scaling value.
4. Both (a) and (b).

**ANS: c.** y **is the scaling value**.

35: srand:

1. Should be called before each call to rand.
2. Should be used instead of rand to generate truly random numbers.
3. Is unnecessary in C++.
4. Can use the timefunction’s return value as an optimal seed value.

**ANS: d. Can use the** time **function’s return value as an optimal seed value**.

**Section 6.9 Storage Classes**

36: In C++0x, Depending on the circumstances, the compiler may ignore the storage class specifier:

1. auto.
2. register.
3. static.
4. extern.

**ANS: b.** register**.**

37: Which of the following is *not* true of static local variables?

1. They are accessible outside of the function in which they are defined.
2. They retain their values when the function in which they are defined terminates.
3. They are initialized to zero if not explicitly initialized by the programmer.
4. They can be of type int.

**ANS: a. They are accessible outside of the function in which they are defined.**

38: What happens when two blocks, one nested inside of the other, both declare variables with the same identifier? (Assume that the outer block declares its variable before the opening left-brace of the inner block.)

1. A syntax error occurs.
2. The “outer” variable is hidden while the “inner” variable is in scope.
3. The “outer” variable is irretrievably lost when the “inner” variable is declared.
4. The “inner” declaration is ignored and the “outer” variable has scope even inside the inner block.

**ANS: b. The “outer” variable is hidden while the “inner” variable is in scope**.

**Section 6.11 Function Call Stack and Activation Records**

39: An activation record will be *popped off* the function call stack whenever:

1. A function returns control to its caller.
2. A function calls another function.
3. A function calls itself.
4. A function declares a local variable.

**ANS: a. A function returns control to its caller.**

40: Which of the following is *not* included in a function’s activation record?

1. The return address of its caller function.
2. Parameter values received from its caller.
3. Local variables it has declared.
4. The name of the function.

**ANS: d. The name of the function**.

41: The inline keyword:

1. Increases function-call overhead.
2. Can reduce a function’s execution time but increase program size.
3. Can decrease program size but increase the function’s execution time.
4. Should be used with all frequently used functions.

**ANS: b. Can reduce a function’s execution time but increase program size.**

**Section 6.14 References and Reference Parameters**

42: When an argument is passed-by-value, changes in the called function \_\_\_\_\_\_\_\_\_\_ affect the original variable’s value; when an argument is passed call-by-reference, changes in the called function \_\_\_\_\_\_\_\_\_\_ affect the original variable’s value.

1. Do not, do.
2. Do not, do not.
3. Do, do.
4. Do, do not.

**ANS: a. Do not, do.**

43: A reference parameter:

1. Is an alias for its corresponding argument.
2. Is declared by following the parameter’s type in the function prototype by an ampersand (&).
3. Cannot be modified.
4. Both (a) and (b).

**ANS: d. Both (a) and (b).**

44: Call-by-reference can achieve the security of call-by-value when:

1. The value being passed is small.
2. A large argument is passed in order to improve performance.
3. A pointer to the argument is used.
4. The const qualifier is used.

**ANS: d. The** const **qualifier is used.**

45: Recursion is memory-intensive because:

1. Recursive functions tend to declare many local variables.
2. Previous function calls are still open when the function calls itself and the activation records of these previous calls still occupy space on the call stack.
3. Many copies of the function code are created.
4. It requires large data values.

**ANS: b. Previous function calls are still open when the function calls itself and the activation records of these previous calls still occupy space on the call stack.**

46: An array is *not*:

1. A consecutive group of memory locations.
2. Subscripted by integers.
3. Declared using braces, [].
4. Made up of different data types.

**ANS d. Made up of different data types.**

47: Which of the following is *false*?

1. The last element of an array has position number one less than the array size.
2. The position number contained within square brackets is called a subscript.
3. A subscript cannot be an expression.
4. All of the above.

**ANS c. A subscript cannot be an expression.**

**Section 7.3 Declaring Arrays**

48: Which statement would be used to declare a 10-element integer array c?

1. array c = int[ 10 ];
2. c = int[ 10 ];
3. int array c[ 10 ];
4. int c[ 10 ];

**ANS d.** int c[ 10 ];

**Section 7.4 Examples Using Arrays**

49: Which of the following is *not* a correct way to initialize an array?

1. int n[ 5 ] = { 0, 7, 0, 3, 8, 2 };
2. int n[] = { 0, 7, 0, 3, 8, 2 };
3. int n[ 5 ] = { 7 };
4. int n[ 5 ] = { 9, 1, 9 };

**ANS: a.** int n[ 5 ] = { 0, 7, 0, 3, 8, 2 };

50: Which of the following is *false* about a function to which an array is being passed?

1. It always knows the size of the array that is being passed.
2. It is being passed the address of the first element in the array.
3. It is able to modify the values stored in the array.
4. The array name is used as an argument in the function call.

**ANS: a. It always knows the size of the array that is being passed.**

**Section 8.2 Pointer Variable Declarations and Initialization**

51: Pointers *cannot* be used to:

1. Contain memory addresses.
2. Reference values directly.
3. Pass an argument by reference.
4. Manipulate dynamic data structures.

**ANS: b. Reference values directly.**

52: Pointers may be assigned which of the following values?

1. Any integer values.
2. An address.
3. NULL.
4. Both (b) and (c).

**ANS d. Both (b) and (c).**

53: What does the following statement declare?

int \*countPtr, count;

1. Two int variables.
2. One pointer to an int and one int variable.
3. Two pointers to ints.
4. The declaration is invalid.

**ANS: b. One pointer to an** int **and one** int **variable.**

54: Three of the following expressions have the same value. Which of the following expressions has a value *different* from the others’?

1. \*&ptr
2. &\*ptr
3. \*ptr
4. ptr

**ANS c.** \*ptr

55: Which of the following is *not* a valid way to pass arguments to a function in C++?

1. By reference with reference arguments.
2. By value.
3. By reference with pointer arguments.
4. By value with pointer arguments.

**ANS: d. By value with pointer arguments.**

56: When a compiler encounters a function parameter for a single-subscripted array of the form int a[], it converts the parameter to:

1. int a
2. int &a
3. int \*a
4. No conversion is necessary.

**ANS c.** int \* a

**Section 8.5 Using** const **with Pointers**

57: A function that modifies an array by using pointer arithmetic such as ++ptr to process every value should have a parameter that is:

1. A nonconstant pointer to nonconstant data.
2. A nonconstant pointer to constant data.
3. A constant pointer to nonconstant data.
4. A constant pointer to constant data.

**ANS: a. A nonconstant pointer to nonconstant data.**

58: A function that prints a string by using pointer arithmetic such as ++ptr to output each character should have a parameter that is:

1. A nonconstant pointer to nonconstant data.
2. A nonconstant pointer to constant data.
3. A constant pointer to nonconstant data.
4. A constant pointer to constant data.

**ANS: b. A nonconstant pointer to constant data.**

59: Which of the following best describes the array name n in the declaration int n[10];?

1. n is a nonconstant pointer to nonconstant data.
2. n is a nonconstant pointer to constant data.
3. n is a constant pointer to nonconstant data.
4. n is a constant pointer to constant data.

**ANS: c. n is a constant pointer to nonconstant data.**

60: What method should be used to pass an array to a function that does not modify the array and only looks at it using array subscript notation:

1. A nonconstant pointer to nonconstant data.
2. A nonconstant pointer to constant data.
3. A constant pointer to nonconstant data.
4. A constant pointer to constant data.

**ANS: d. A constant pointer to constant data.**

61: The correct function name for overloading the addition (+) operator is:

1. operator+
2. operator(+)
3. operator:+
4. operator\_+

**ANS a.** operator+

62: Which of the following operators *cannot* be overloaded?

1. The . operator.
2. The -> operator.
3. The & operator.
4. The [ ] operator.

**ANS a. The** . **operator.**

63 Which statement about operator overloading is *false*?

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

**ANS b. Certain overloaded operators can change the number of arguments they take.**

64 To implicitly overload the += operator:

1. Only the + operator needs to be overloaded.
2. Only the = operator needs to be overloaded.
3. Both the + and = operators need to be overloaded.
4. The += operator cannot be overloaded implicitly.

**ANS d. The** += **operator cannot be overloaded implicitly.**