Final Review Fall 2015

MC

Pointers

1. Pointers *cannot* be used to:

a. Contain memory addresses.
b. Reference values directly.
c. Pass an argument by reference.
d. Manipulate dynamic data structures.

2. Pointers may be assigned which of the following values?

a. Any integer values.
b. An address.
c. `nullptr`.
d. Both (b) and (c).

3. What does the following statement declare?

```
int *countPtr, count;
```

a. Two `int` variables.
b. One pointer to an `int` and one `int` variable.
c. Two pointers to `int`s.
d. The declaration is invalid.

4. All of the following *can* cause a fatal **execution-time** error *except*:

a. Dereferencing a pointer that has not been assigned to point to a specific address.
b. Dereferencing a pointer that has not been initialized properly.
c. Dereferencing a null pointer.
d. Dereferencing a variable that is not a pointer.

5. Three of the following expressions have the same value. Which of the following expressions has a value *different* from the others'?

a. `*&ptr`
b. `&*ptr`
c. `*ptr`
d. `ptr`

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

a. By reference with reference arguments.
b. By value.
c. By reference with pointer arguments.
d. By value with pointer arguments.

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

a. `int a`
b. `int &a`
c. `int *a`
d. No conversion is necessary.

8. Which statement would be used to declare a 10-element integer array `c`?

a. `array c = int[ 10 ];`
b. `c = int[ 10 ];`
c. `int array c[ 10 ];`
d. `int c[ 10 ];`

9. To *prevent* modification of a built-in array's values when you pass the built-in array to a function:

a. The built-in array must be declared `static` in the function.
b. The built-in array parameter can be preceded by the `const` qualifier.
c. A copy of the built-in array must be made inside the function.
d. The built-in array must be passed by reference.

10. Which of the following is *false* about a function to which a built-in array is being passed?

a. It always knows the size of the built-in array that is being passed.
b. It is being passed the address of the first element in the built-in array.
c. It is able to modify the values stored in the built-in array.
d. The built-in array's name is used as an argument in the function call.

11. Given a *built-in array* of `int`s named `values`, which of the following statements would sort the array?

a. `sort( values.begin(), values.end() );`
b. `sort( values.array_begin(), values.array_end() );`
c. `sort( begin( values ), end( values ) );`
d. `sort( array_begin( values ), array_end( values ) );`

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

a. A nonconstant pointer to nonconstant data.
b. A nonconstant pointer to constant data.
c. A constant pointer to nonconstant data.
d. A constant pointer to constant data.

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

a. A nonconstant pointer to nonconstant data.
b. A nonconstant pointer to constant data.
c. A constant pointer to nonconstant data.
d. A constant pointer to constant data.

14. Which of the following best describes the array name `n` in the declaration `int n[10];`?

a. `n` is a nonconstant pointer to nonconstant data.
b. `n` is a nonconstant pointer to constant data.
c. `n` is a constant pointer to nonconstant data.
d. `n` is a constant pointer to constant data.

15. 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:

a. A nonconstant pointer to nonconstant data.
b. A nonconstant pointer to constant data.
c. A constant pointer to nonconstant data.
d. A constant pointer to constant data.

16. `sizeof`:

a. Is a binary operator.
b. Returns the total number of elements in an array.
c. Usually returns a `double`.
d. Returns the total number of bytes in a variable.

17. Which of the following gives the number of elements in the array `int r[ 10 ]`?

a. `sizeof r`
b. `sizeof ( *r )`
c. `sizeof r / sizeof ( int )`
d. `sizeof ( *r ) / sizeof ( int )`

18. Which of the following *can* have a pointer as an operand?

a. `++`
b. `*=`
c. `%`
d. `/`

19. Given that `k` is an integer array starting at location 2000, `kPtr` is a pointer to `k` and each integer is stored in 4 bytes of memory, what location does `kPtr + 3` point to?

a. 2003
b. 2006
c. 2012
d. 2024

20. A pointer *can not* be assigned to:

a. Another pointer of the same type without using the cast operator.
b. A pointer to `void` without using the cast operator.
c. A pointer of a type other than its own type and `void` *without* using the cast operator.
d. Any other pointer by using the cast operator.

21. Comparing pointers and performing pointer arithmetic on them is meaningless unless:

a. They point to elements of the same array.
b. You are trying to compare and perform pointer arithmetic on the values to which they point.
c. They point to arrays of equal size.
d. They point to arrays of the same type.

22. Assuming that `t` is an array and `tPtr` is a pointer to that array, which expression refers to the address of element 3 of the array?

a. `*( tPtr + 3 )`
b. `tPtr[ 3 ]`
c. `&t[ 3 ]`
d. `*( t + 3 )`

23. Which of the following is *false* for pointer-based strings?

a. A string may include letters, digits and various special characters (i.e., +, -, *).
b. A string in C++ is an array of characters ending in the null character (`'\0'`).
c. String literals are written inside of single quotes.
d. A string may be assigned in a declaration to either a character array or a variable of type `char *`.

`24. cin.getline( superstring, 30 );`
is equivalent to which of the following?

a. `cin.getline( superstring, 30, '\0' );`
b. `cin.getline( superstring, 30, '\n' );`
c. `cin.getline( superstring, 30, '\s' );`
d. `cin.getline( superstring, 30, '\t' );`

25. A string array:

a. Stores an actual string in each of its elements.
b. Can only provide access to strings of a certain length.
c. Is actually an array of pointers.
d. Is always less memory efficient than an equivalent double-subscripted array.

26. A string array is commonly used for:

a. Command-line arguments.
b. Storing an extremely long string.
c. Storing multiple copies of the same string.
d. Displaying floating-point numbers to the screen.

27. Which of the following is *not* true of pointers to functions?

a. They contain the starting address of the function code.
b. They are dereferenced in order to call the function.
c. They can be stored in arrays.
d. They can not be assigned to other function pointers.

28. `( *max )( num1, num2, num3 );`:
    a. Is the header for function `max`.
    b. Is a call to the function pointed to by `max`.
    c. Is the prototype for function `max`.
    d. Is a declaration of a pointer to a function called `max`.

Operator Overloading

29. Which statement about operator overloading is *false*?

    a. Operator overloading is the process of enabling C++'s operators to work with class objects.

    b. C++ overloads the addition operator (+) and the subtraction operator (-) to perform differently, depending on their context in integer, floating-point and pointer arithmetic with data of fundamental types.

    c. You can overload all C++ operators to be used with class objects.

    d. When you overload operators to be used with class objects, the compiler generates the appropriate code based on the types of the operands.

30. Which of the following is *false*?

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

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

    a. `operator+`
    b. `operator(+)`
    c. `operator:+`
    d. `operator_+`

32. Which of the following operators *cannot* be overloaded?

    a. The `.` operator.
    b. The `->` operator.
    c. The `&` operator.
    d. The `[ ]` operator.

33. Which statement about operator overloading is *false*?

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

34. Which situation would *require* the operator to be overloaded as a non-member function?

a. The overloaded operator is `=`.
b. The left most operand must be a class object (or a reference to a class object).
c. The left operand is an `int`.
d. The operator returns a reference.

35. 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:

a. `ostream &operator<<( ostream &output, const Data &dataToPrint )`
b. `ostream operator<<( ostream &output, const Data &dataToPrint )`
c. `ostream &operator<<( const Data &dataToPrint, ostream &output )`
d. `ostream operator<<( const Data &dataToPrint, ostream &output )`

36. The conventional way to distinguish between the overloaded preincrement and postincrement operators (`++`) is:

a. To assign a dummy value to preincrement.
b. To make the argument list of postincrement include an `int`.
c. To have the postincrement operator call the preincrement operator.
d. Implicitly done by the compiler.

37. Because the postfix increment operator returns objects by value and the prefix increment operator returns objects by reference:

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

38. 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:

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

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

   Date temp = *this;
   Increment();
   return temp;
}
```

Classes and inheritance

39. Member access specifiers (`public` and `private`) can appear:
    a. In any order and multiple times.
    b. In any order (`public` first or `private` first) but not multiple times.
    c. In any order and multiple times, if they have brackets separating each type.
    d. Outside a class definition.

40. Which of the following preprocessor directives does *not* constitute part of the preprocessor wrapper?
    a. `#define`
    b. `#endif`
    c. `#ifndef`
    d. `#include`

41. Member function definitions:
    a. Always require the scope resolution operator (`::`).
    b. Require the scope resolution operator only when being defined outside of the definition of their class.
    c. Can use the scope resolution operator anywhere, but become `public` functions.
    d. Must use the scope resolution operator in their function prototype.

42. Which of the following is *not* a kind of inheritance in C++?
    a. `public`.
    b. `private`.
    c. `static`.
    d. `protected`.

43. The *is-a* relationship represents.
    a. Composition.
    b. Inheritance.
    c. Information Hiding.
    d. A friend.

44. To declare class `subClass` a `private`ly derived class of `superClass` one would write:
```
a. class subclass : private superClass
b. class subclass :: private superClass
c. class subclass < private superClass >
d. class subclass inherits private superClass
```

45. Assuming the following is the beginning of the constructor definition for class `BasePlus-CommissionEmployee` which inherits from class `Point`,
```
BasePlusCommissionEmployee::BasePlusCommissionEmployee( string first,
   string last, string ssn, double sales, double rate, double salary )
   : CommissionEmployee( first, last, ssn, sales, rate )
```
The line after the colon (:) will
    a. Invokes the `CommissionEmployee` constructor with arguments.
    b. Causes a compiler error.
    c. Is unnecessary because the `CommissionEmployee` constructor is called automatically.
    d. Indicates inheritance.

Polymorphism

46. Polymorphism is implemented via:

a. Member functions.
b. `virtual` functions and dynamic binding.
c. `inline` functions.
d. Non-`virtual` functions.

47. Which of the following would not be a member function that derived classes `Fish`, `Frog` and `Bird` should inherit from base class `Animal` and then provide their own definitions for, so that the function call can be performed polymorphically?

a. `eat`
b. `sleep`
c. `move`
d. `flapWings`

48. `Employee` is a base class and `HourlyWorker` is a derived class, with a redefined non-`virtual` `print` function. Given the following statements, will the output of the two `print` function calls be identical?

```
HourlyWorker h;
Employee *ePtr = &h;

ePtr->print();
ePtr->Employee::print();
```

a. Yes.
b. Yes, if `print` is a `static` function.
c. No.
d. It would depend on the implementation of the `print` function.

49. If objects of all the classes derived from the same base class all need to draw themselves, the `draw` function would most likely be declared:

a. `private`
b. `virtual`
c. `protected`
d. `friend`

50. Which of the following statements is *true*?

a. In C++11, all classes can be used as base classes.
b. In C++11, only classes that are not declared as final can be used as base classes.
c. In C++11, only classes that are declared as base can be used as base classes.
d. None of the above

51. The line:

```
virtual double earnings() const = 0;
```

appears in a class definition. You *cannot* deduce that:

a. All classes that directly inherit from this class will *override* this method.
b. This class is an abstract class.
c. Any concrete class derived from this class will have an `earnings` function.
d. This class will probably be used as a base class for other classes.

Files and Streams

52. Which of the following does *not* have a stream associated with it?
    a. `cerr`.
    b. `cin`.
    c. `cout`.
    d. All of the above have streams associated with them.

53. In order to perform file processing in C++, which header files *must* be included?
    a. `<cstdio>` , `<iostream>` and `<fstream>`.
    b. `<cstdio>` and `<iostream>`.
    c. `<cstdio>` and `<fstream>`.
    d. `<iostream>` and `<fstream>`.

54. Select the *false* statement.
    a. C++ imposes no structure on a file.
    b. C++ files include information about their structure.
    c. The programmer must impose a structure on a file.
    d. C++ files do not understand notions such as "records" and "fields."

55. Which file open mode would be used to write data only at the end of an existing file?
    a. `ios::app`
    b. `ios::in`
    c. `ios::out`
    d. `ios::trunc`

56. [C++11]: Which of the following statements is *true*?
    a. When opening a file, you can specify the name of the file only as a pointer-based string.
    b. When opening a file, you can specify the name of the file only as a `string` object.
    c. When opening a file, you can specify the name of the file as either a pointer-based string or a `string` object.
    d. None of the above.

57. Random access files are *more* effective than sequential files for:
    a. Instant access to data.
    b. Updating data easily.
    c. Inserting data into the file without destroying other data.
    d. All of the above.

58. A random access file is organized *most* like a(n):
    a. Array.
    b. Object.
    c. Class.
    d. Pointer.

59. Select the proper object type.

```
__________ file( "file.dat", ios::in | ios::out );
```

a. `iostream`
b. `fstream`
c. `ofstream`
d. `ifstream`

60. Select the *correct* statement regarding C++ I/O streams:

a. C++ provides only high-level I/O capabilities because it is a high-level programming language.
b. High-level (formatted) I/O is best for large-volume transfers.
c. Low-level I/O breaks information down into small, meaningful groups of related bytes.
d. Programmers generally prefer high-level I/O to low-level I/O.

61. _____ is usually *faster* than _____.

a. High-level I/O, low-level I/O.
b. Low-level I/O, high-level I/O.
c. Low-level I/O, internal data processing.
d. High-level I/O, internal data processing.

62. Which C++ data type was designed to store Unicode characters?

a. `char`
b. `long`
c. `wchar_t`
d. `size_t`

63. Which of the following classes is a *base class* of the other three?

a. `basic_ios`
b. `basic_istream`
c. `basic_ostream`
d. `basic_iostream`

64. Which of the following is *not* an object of the `ostream` class?

a. `cout`
b. `cerr`
c. `cin`
d. `clog`

65. Which of the following is *not* a member function of the C++ `ostream` class?

a. Stream-insertion operator (<<).
b. Stream-extraction operator (>>).
c. `put`.
d. `write`.

66. Which of the following prints the address of character string `string` given the following declaration?

```
char * string = "test";
```

a. `cout << string;`
b. `cout << *&string;`
c. `cout << static_cast< void * >( string );`
d. `cout << * string;`

67. Which of the following is an *illegal* use of function `put`?
    a. `cout.put( 'A' );`
    b. `cout.put( "A" );`
    c. `cout.put( 'A' ).put( '\n' );`
    d. `cout.put( 65 );`

68. Which of the following is *not* true about `setw` and `width`?
    a. If the width set is *not* sufficient the output prints as wide as it needs.
    b. They are used to set the field width of output.
    c. Both of them can perform two tasks, setting the field width and returning the current field width.
    d. They only apply for the next insertion/extraction.

69. Which of the following stream manipulators causes an outputted number's sign to be *left justified*, its magnitude to be *right justified* and the center space to be filled with fill characters?
    a. `left`
    b. `right`
    c. `internal`
    d. `showpos`

70. Which of the following statements restores the *default fill character*?
    a. `cout.defaultFill();`
    b. `cout.fill();`
    c. `cout.fill( 0 );`
    d. `cout.fill( ' ' );`

71. What will be *output* by the following statements?

```
double x = .0012345;
cout << fixed << x << endl;
cout << scientific << x << endl;
```

    a. 1.234500e-003
       0.001235
    b. 1.23450e-003
       0.00123450
    c. .001235
       1.234500e-003
    d. 0.00123450
       1.23450e-003

Standard Template Library

72. Which of the following is *not* a key component of the Standard Library?
    a. Containers.
    b. Iterators.
    c. Algorithms.
    d. Pointers.

73. Which of the following containers is *not* considered a near container?
    a. C-like arrays

b. `vectors`
c. `strings`
d. `bitsets`

74. Iterators are similar to pointers because of the:
a. * and ++ operators.
b. -> operator.
c. `begin` and `end` functions.
d. & operator.

75. Which of the following is the correct hierarchy of iterator categories (*weakest* at the *left*)?
a. Input/output, forward, bidirectional, random access.
b. Random access, forward, bidirectional, input/output.
c. Bidirectional, forward, random access, input/output.
d. Input/output, bidirectional, forward, random access.

76. A Standard Library algorithm *cannot*:
a. Return an iterator.
b. Take two iterators as arguments to specify a range.
c. Access Standard Library members directly.
d. Be used with containers that support more powerful iterators than the minimum requirements for the algorithm.

77. Which of the following is *not* a sequence container provided by the Standard Library?
a. `vector`
b. `array`
c. `list`
d. `deque`

78. Which of the following applications would a deque *not* be well suited for?
a. Applications that require frequent insertions and deletions at the front of a container.
b. Applications that require frequent insertions and deletions in the middle of a container.
c. Applications that require frequent insertions and deletions at the back of a container.
d. Applications that require frequent insertions and deletions at the front and at the back of a container.

79. Which of the following is *not* a member function of all sequence containers?
a. `front`
b. `middle`
c. `back`
d. `pop_back`

80. Which of the following is a *difference* between `vector`s and arrays?
a. Access to any element using the `[]` operator.
b. Stored in contiguous blocks of memory.
c. The ability to change size dynamically.
d. Efficient direct access to any element.

81. The `erase` member function of class `vector` *cannot*:

a. Specify an element to be removed from the `vector`.
b. Specify a value to be removed from the `vector`.
c. Specify a range of elements to be removed from the `vector`.
d. Be called by member function `clear`.

82. The `list` sequence container does *not*:
a. Efficiently implement `insert` and `delete` operations anywhere in the `list`.
b. Use a doubly linked list.
c. Support bidirectional iterators.
d. Automatically sort inserted items.

83. Class `deque` provides:
a. Efficient indexed access to its elements.
b. The ability to add storage at either end of the `deque`.
c. Efficient insertion and deletion operations at the front and back of a `deque`.
d. All of the above.

84. The main *difference* between `set` and `multiset` is:
a. Their interface.
b. That one deals with keys only, and the other deals with key/value pairs.
c. Their efficiency.
d. How they handle duplicate keys.

85. If a program attempts to insert a duplicate key into a `set`:
a. An exception is thrown.
b. The `set` will contain multiple copies of that key.
c. A compile error will occur.
d. The duplicate key will be ignored.

86. If `pairs` is a `map` containing `int` keys and `double` associated values, the expression `pairs[ 5 ] = 10`:
a. Associates the value `10.0` to the key `5` in `pairs`.
b. Associates the value `5.0` to the key `10` in `pairs`.
c. Associates the value associated with key `10` to key `5` in `pairs`.
d. Associates the value associated with key `5` to key `10` in `pairs`.

87. To pop an element off the top of a `stack` for processing:
a. Use member function `top`.
b. Use member function `pop`.
c. Use member function `top` and then member function `pop`.
d. Use member function `pop` and then member function `top`.

88. Which of the following is a *not* a member function of `queue`?
a. `enqueue`
b. `pop`
c. `empty`
d. `size`

Algorithms

89. The algorithms in the Standard Library:
    a. Use `virtual` function calls.
    b. Are implemented as member functions of the container classes.
    c. Do *not* depend on the implementation details of the containers on which they operate.
    d. Are *not* as efficient as the algorithms presented in most textbooks.

90. The easiest way to set all the values of a `vector` to zero is to use function:
    a. `fill`
    b. `fill_n`
    c. `generate`
    d. `generate_n`

91. Which of the following function calls is a *valid* way to place elements into `vector< char > chars`?
    a. `std::fill( chars.begin(), chars.end(), '5' );`
    b. `std::fill_n( chars.begin(), chars.end(), '5' );`
    c. `std::generate( chars.begin(), 10, '5' );`
    d. `std::generate_n( 10, chars.end(), '5' );`

92. Given that `v1` and `v2` are `vector`s, what is returned by the function call

```
std::equal( v1.begin(), v1.end(), v2.begin() )
```

    a. A `bool` indicating whether `v1` and `v2` are equal.
    b. A `bool` indicating whether the first element of `v1`, the last element of `v1` and the first element of `v2` are all equal.
    c. An iterator pointing to the first location where `v1` and `v2` are equal.
    d. An iterator pointing to the first location where `v1` and `v2` are not equal.

93. Function `mismatch` returns:
    a. The position number where the two specified sequences do *not* match.
    b. A `pair` containing the two elements in the specified sequences that do *not* match.
    c. A `pair` containing two iterators pointing to the two locations in the specified sequences that do *not* match.
    d. A `bool` indicating whether the two specified sequences do *not* match.

94. Which of the following is *not* a mathematical algorithm included in the Standard Library?
    a. `min_element`
    b. `copy`
    c. `transform`
    d. `accumulate`

95. The easiest way to search through a list of names and output the first one that begins with a vowel would be to use function:

    a. `find`
    b. `find_if`
    c. `sort`
    d. `binary_search`

96. Which of the following statements produces the *same* results as the statement:

```
std::copy( v1.begin(), v1.end(), v2.begin() );
```

if v1 and v2 are both 10-element vectors?

a. `std::copy_backward( v1.begin(), v1.end(), v2.begin() );`
b. `std::copy_backward( v2.begin(), v2.end(), v1.begin() );`
c. `std::copy_backward( v1.begin(), v1.end(), v2.end() );`
d. `std::copy_backward( v2.begin(), v2.end(), v1.end() );`

97. The _____ function would produce the sequence 1, 5, 6 when passed the sequences 1, 2, 3, 4, 5, 6 and 2, 3, 4, 7 as first/second and third/fourth arguments, respectively.

a. `set_intersection`
b. `set_difference`
c. `set_union`
d. `set_symmetric_difference`

98. The _____ function would produce a sequence containing three elements when passed the sequences 1, 2 and 1, 2, 3, 4, 5 as first/second and third/fourth arguments, respectively.

a. `set_intersection`
b. `set_difference`
c. `set_union`
d. `set_symmetric_difference`

99. Sorting a preexisting sequence of *n* elements can be accomplished with the *heapsort algorithm* by:

a. Calling `make_heap` on the entire sequence and then calling `pop_heap` on the entire sequence *n* times.
b. Calling `push_heap` on the entire sequence *n* times and then calling `pop_heap` on the entire sequence *n* times.
c. Calling `make_heap` on the entire sequence and then calling `sort_heap` on the entire sequence.
d. Calling `push_heap` on the entire sequence *n* times and then calling `sort_heap` on the entire sequence.

100. Which of the following function calls would *not* return the value that is its *first* argument?

a. `std::min( 3, 23 )`
b. `std::min( 'N', 'P' )`
c. `std::max( 17, 16 )`
d. `std::max( 'd', 'k' )`