CSCI 241 Exam 2 Study Guide

C++ Variable Declarations

- Be able to read and code variable declarations for ordinary variables, arrays, pointers, or references, particularly those using the `const` keyword.

The `const` Keyword

- Be able to declare a pointer to `const` data, a `const` pointer, or a `const` pointer to `const` data. Know what restrictions doing so places on using the pointer.
- Be able to declare a reference to `const` data. Know what restrictions this places on using the reference.
- Know how to declare a method of a class as `const`.
- Know the things that can't be done in a `const` method:
  - Change the data members of the object that called the method
  - Call a non-`const` method for the object that called the method
- An object that is not `const` can call a `const` method or a non-`const` method. An object that is `const` (or a pointer to a `const` object or a reference to a `const` object) can **only** call methods that are `const`.
- Constructors and destructors can never be `const` methods.

Scope

- You should know the three types of scope described in class:
  - *Block scope / local scope* – local variables, function and method parameters. In scope from the point of declaration to the end of the block in which they are declared.
  - *File scope / global scope* – global variables, standalone functions. In scope from the point of declaration to the end of the file in which they are declared.
  - *Class scope* – data members and methods of a class. In scope in all methods of the class.
- You should understand the concept of shadowing, the situations in which it may occur, and the techniques to resolve those situations.

Storage Class

- Know the three types of storage described in class:
  - Automatic storage – default for block scope variables. Allocated and initialized at the point where the variable is declared; de-allocated when the variable goes out of scope.
  - Static storage – default for file scope variables. Allocated and initialized at the beginning of the program; de-allocated at the end of the program.
  - Dynamic storage – allocated at runtime using `new` / `new[]`; de-allocated by `delete` / `delete[]` or at the end of the program.
- Know how to change the storage class of a local variable to static by putting the `static` keyword on its declaration.

Linkage

- You should know the three types of linkage described in class:
  - *No linkage* – local variables, function and method parameters. May not have their scope extended to a different source file.

- *Internal linkage* – global variables or standalone functions with the `static` keyword on their declaration or prototype. May not have their scope extended to a different source file.
- *External linkage* – global variables or standalone functions without the `static` keyword on their declaration or prototype. May have their scope extended to a different source file. For a function, place the function's prototype in the other source file. For a global variable, place an `extern` declaration in the other source file.

Default Arguments

- Default values for method or function parameters may be coded as part of a prototype.
- Parameters with default values must be the trailing parameters in the function prototype parameter list.
- When a function defined with default parameter values is called with trailing arguments missing, the default values are used.

Function and Method Overloading

- You should know the criteria used by the compiler to distinguish between two or more functions or methods with the same name and in the same scope:
  - The number of arguments
  - The data types of the arguments
  - The order of the data types
  - Whether or not a method is `const`
- The return data type of the method is not one of the criteria used.

The `this` Pointer

- The `this` pointer points to the object that called a method.
- For a method of class `ClassName`, the data type of the `this` pointer is either `ClassName*` (if the method is not `const`) or `const ClassName*` (for a `const` method).
- Standalone functions do not have a `this` pointer since they are not called for an object.

The `friend` Keyword

- Know how to declare a class or standalone function to be a friend of a class.
- Friendship grants direct access to the `private` members of a class.
- Friendship must always be explicitly declared:
  - If A is a friend of B, B is not automatically a friend of A
  - If A is a friend of B and B is a friend of C, A is not automatically a friend of C

Operator Overloading

- Be able to list the aspects of an operator that cannot be changed by operator overloading – precedence, number of arguments, direction of evaluation, how the operator works with built-in types.
- Know which operators must be overloaded as methods, and when an operator must be overloaded as a standalone function.
- Know what the method or function call generated by the compiler for an overloaded operator will look like. Which operand will call the method? Which operand or operands will be passed as arguments to the method or function?
- Be able to write overloaded operator functions similar to those used on programming assignments – stream insertion operator, relational operators, arithmetic operators, subscript operator.

Dynamic Storage Allocation

- Know how to use the `new[]` operator to allocate dynamic storage for an array.
- A dynamically-allocated array of objects created with `new[]` will have a constructor called for each object of the array.
- Know how to use the `delete[]` operator to de-allocate a dynamic array.
- A dynamically-allocated array of objects will have the destructor called for each object of the array when it is deleted with `delete[]`.
- A "shallow" copy of an object copies only the object but not the dynamic storage that it owns. A "deep" copy of an object copies the object and the dynamic storage that it owns.
- A class that allocates dynamic storage for one or more of its data members requires coding all three of following methods:
    - Destructor
    - Copy constructor
    - Copy assignment operator

Destructor

- A destructor is called for a class object when it goes out of scope, is deleted, or the program ends.
- Be able to write a destructor for a class that dynamically created an array.

Copy Constructor

- Be able to list the three situations which can result in a call to the copy constructor:
    1. When an object is declared and initialized with another object of the same class
    2. When an object is passed by value
    3. When an object is returned by value
- Know how to write a copy constructor for a class that dynamically creates an array.

Copy Assignment Operator

- Know how to write a copy assignment operator for a class that dynamically creates an array.

Abstract Data Type Definition

- An *abstract data type* is a data type defined in terms of what data may be stored and the operations that may be performed on it. It does not specify **how** the data is represented in memory.

Stack and Queue ADTs

- Know the type of data structure category that a stack or queue falls into (Last In, First Out vs. First In, First Out)
- Know the types of errors that can occur when using a stack or queue (underflow on `pop()` or `front()`, etc.)
- Be able to add an item to a stack or queue (array implementation only).
- Be able to remove an item from a stack or queue (array implementation only).
- Be familiar with the other typical operations performed on a stack or queue (`size()`, `empty()`, copy constructor, copy assignment operator, etc.)

Sample Exam Questions

1. Only an object declared as `const` can call a `const` method.

   A. true.
   B. false.

2. When a class uses dynamically-allocated storage, which additional methods should be implemented?

   A. Copy assignment operator.
   B. Copy constructor.
   C. Destructor.
   D. All of the above.

3. Which of the following declares the variable `p` to be a constant pointer to a `char`?

   A. `const char* p;`
   B. `const char p;`
   C. `char* const p;`
   D. `char* p const;`

4. What is the variable `this`?

   A. A C++ reference to the object that called a method.
   B. A C++ pointer to the object that called a method.
   C. A C++ pointer to an array of strings.
   D. A C++ pointer to a method.

5. A local variable declared inside an `if` statement has

   A. function scope
   B. block scope
   C. file scope
   D. class scope

6. When overloading a C++ method, which of the following criteria is NOT used to distinguish between different versions of the method?

   A. the number of arguments the method takes
   B. whether or not the method is `const`
   C. the data types of the arguments for the method
   D. the return data type of the method

7. The stream insertion operator (<<) must be overloaded as a method of a class, not a friend function.

   A. true.
   B. false.

8. Which of the following method prototypes is legal?

   A. `Date(int = 0, int, int = 0);`
   B. `Date(int = 0, int, int);`
   C. `Date(int, int = 0, int = 0);`
   D. `Date(int, int = 0, int);`

9. Suppose that a new `Matrix` class that represents a matrix has the following prototype for an overloaded addition operator method that adds two matrices:

```
Matrix operator*(const Matrix&) const;
```

   For the expression `m1 * m2` (where `m1` and `m2` are both objects of the class `Matrix`), what will be the actual method call generated by the compiler?

   A. `operator*(m1, m2);`
   B. `m2.operator*(m1);`
   C. `m1.operator*(m2);`
   D. Overloaded operators never generate method calls.

10. If the statement `friend class A;` appears inside the declaration of `class B`, and the statement `friend class B;` appears inside the declaration of `class C` then

    A. `class A` is a friend of `class C`.
    B. `class A` can access `private` members and methods of `class B`.
    C. `class C` can access `private` members and methods of `class A`.
    D. `class B` can access `private` members and methods of `class A`.

11. A global variable has what kind of linkage by default?

    A. internal linkage.
    B. external linkage.
    C. no linkage.
    D. C++ variables don't have linkage.

12. Which of the following C++ identifiers have *block scope*?

    A. Methods of a class and data members of a class.
    B. Local variables and arguments declared in a class method.
    C. Global variables and standalone functions.
    D. Both A and B.

13. When a C++ operator is overloaded, we can change the number of operands that the operator takes.

    A. true.
    B. false.

14. Assume that you have coded the following C++ statement:

```
Student* studentList = new Student[20];
```

How many times will a `Student` constructor be called as a result of this statement?

15. For a class named `Vector3D`, write the `friend` declaration for a standalone function (NOT a method) to overload the `*` operator. The function should take a `float` as its first argument and a reference to a constant `Vector3D` object as its second argument, and should return a `Vector3D` object.

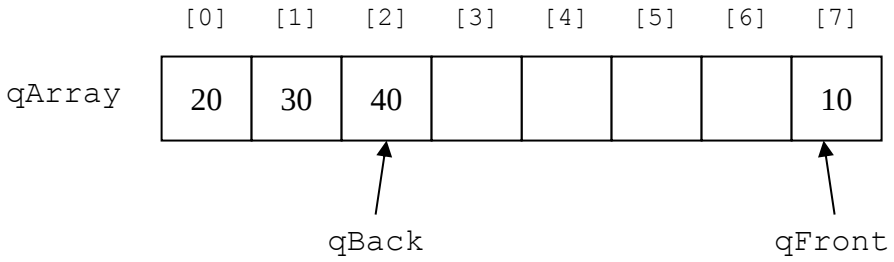
16. Write the two **method headers** (the first line of the method definition, not the prototype) for overloading the `[]` operator for a class called `Triangle`. The data that will be accessed by this operator is an array data member declared as follows:

```
Point points[3];
```

17. List two situations in which a class's copy constructor might be called.

18 What is the difference between a "shallow copy" and a "deep copy" of an object?

19. (6 points) The object `q` is an instance of an array-based queue class. The diagram below shows the current contents of the array of integers used to store items in this queue. The values of the object `q`'s data members are as follows: `qFront` is 7, `qBack` is 2, `qCapacity` is 8, and `qSize` is 4.



Redraw the array and its contents following the execution of the following five method calls. Make sure to indicate which element of the array is the front of the queue and which element is the back. Remember that the order the elements are stored in the queue will not change and existing elements will not move as a result of calls to `push()` and `pop()`.

```
q.push(50);
q.pop();
q.push(60);
q.push(70);
```

20. In a stack, new items are inserted at the _____ of the stack and removed from the _____ of the stack. This means that the _____ item inserted will be the _____ item to be removed.

21. In a queue, new items are inserted at the _____ of the queue and removed from the _____ of the queue. This means that the _____ item inserted will be the _____ item to be removed.

22. Write a method definition for the `size()` method of the `Stack` class shown on the last page.

22. Write a method definition for the `empty()` method of the `Stack` class shown on the last page.

24. Write a method definition for the `push()` method of the `Stack` class shown on the last page. If the dynamic array is full, call the `reserve()` method to increase the capacity in the same way it was done on Assignment 5.

25. Write a method definition for the copy assignment operator of the `Stack` class shown on the last page.

Class declaration for the array-based `Stack` class used in Questions 22 – 25:

```
class Stack
   {
   public:

      Stack();                              // Default constructor
      Stack(const Stack&);                  // Copy constructor
      ~Stack();                             // Destructor
      Stack& operator=(const Stack&);       // Copy assignment operator

      void clear();                  // Sets the stack back to empty
      size_t size() const;           // Returns number of items in stack
      size_t capacity() const;       // Returns capacity of stack array
      bool empty() const;            // True if stack is empty

      void push(int);                // Insert item at top of stack
      void pop();                    // Remove top item
      int top() const;               // Return top item

   private:

      int* stkArray;                 // Pointer to dynamic storage
      size_t stkCapacity;            // Number of array elements
      size_t stkSize;                // Number of items stored in stack

      void reserve(size_t);          // Reserve additional array elements
   };
```