Exam 2 Review

Matt Warner

1 OOP basics (GOOD)

- Given a description of a class, be able to write a header file containing a declaration for that class, including header guards, data member declarations, friend function declarations, and member function prototypes.
- Be able to wrtie common types of member functions such as **default constructor that takes no args**, **a constructor that takes args, accessor and mutator member functions (get and set functions)**.

```
#ifndef HEADER_H
#define HEADER_H

class Entity {

  // Data member declarations
  int x, y;

  // friend function declaration
  friend std::ostream& operator<<(std::ostream&, const Entity&);

  // Constructors
  Entity();

  Entity(int x, int y)
  Entity(const Entity&); // copy constructor (used with pointers)

  // Member functions
  void push(int value);
  void pop();

  // Accessor and mutator functions (get and set)
  void set_x(int x);
  void get_x() const;
};
```

2 Binary Search Algorithm (GOOD)

- Be able to trace the execution of the binary search algorithm as it searches an array for a particular search key. You should be able to track the values of the subscripts **low, high, mid** as the algo progresses.

3 C++ Pointers and References

- Be able to declare a pointer, a pointer to constant data, a constant pointer, or a constant pointer to constant data.

```
// regular pointer
int* ptr = nullptr
int* ptr = &num;

// pointer to constant data
int* const ptr = nullptr;

// const pointer
const int* ptr;
```

```
// const pointer to const data.
const int* const ptr;
```

- Be able to answer questions about pointer syntax, the indirection operator (unary *), the "address of" operator (unary &), and the relationship between a pointer and the variables it points to.

4 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.

- int* const ptr (pointer to const data)
  - pointer to const data cannot change the value stored at the address being pointed to.
- const int* ptr (const pointer)
  - Cannot change the address stored in the pointer variable
- const int* const ptr; (const pointer to const data)
  - this prevents you from changing the value stored at the address being pointed to and it prevents you from changing the address stored in the pointer variable.

- Be able to declare a reference to const data. Know what restrictions this places on using the reference.
  - const string& s (Cannot change the value of the variable that the reference var refers to)
- Be able to list the things that can't be done in a const member function.
  - Cannot change the values of data members
  - cannot call non const methods.
  - an object that is not const can call a const member function or a non-const member function. AN object that is const (or a pointer to a const object or a reference to a const object) can only call member functions that are const.

5 Default Function Arguments

- Default values for function and member function parameters may be coded as part of a prototype.
- Parameters with default values must be 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.

6 Function and Member function overloading

- You should know the criteria used by the compiler to distinguish between two or more functions or member functiosn with the same name and in the same scope
  - The number of args
  - The data types of the args
  - The order of the data types
  - Whether or not a member function is const

7 The this→ pointer

- The this pointer points to the object that called the method.
- For a member function of class *class_name*, the data type of the this pointer is either class_name* (if the member funciton is not const) or const class_name* (for a const member function)
- Standalone functions and static member functions do not have a this pointer since they are not called for a specific object.

8 The friend Keyword

- Know how to declare a class or standalone function to be a friend class.
  - For a class, code the keyword friend followed by the class name.
  - For a function, code the keyword friend followed by the function's prototype
- Friendship grants direct access to the private members of a class.
- Friendship must always be explicity 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 (no transitive property applies)

9 Operator Overloading

- Be able to list the aspects of an operator that cannot be changed by operator overloading
  - Precedence
  - number of arguments (operands)
  - direction evaluation
  - and how the operator works with built-in data types
- Know which operators must be overloaded as member functions, and when an operator must be overloaded as a standalone function
  - The stream insertion operator (" « ")
  - any other function that has an lhs operand that is not a object of our class.
- Know what the function call generated by the compiler for an overloaded operator will look like.
  - a.operator+(b) (this is what it looks like for member functions)
  - operator+(a, b); (this is what it looks like for non-member functions)
- Be able to write overloaded operator functions similar to those used on programming assignments and recitation projects - stream insertion operator, relational operators, arithmetic operators, subscript operators.

10 Dynamic Storage Allocation

- Know how to use the new[] operator to allocate dynamic storage for an array.
- A dynamically-allocated array of objects create with new[] will call the class's default constructor 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 the following functions.
    - destructor
    - copy constructor
    - copy assignment operator

11 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.

12 copy constructor

- Be able to list the three situations which may result in a call to the copy constructor.
    - When a new object is created and initialized with an exisiting object.
    - When an object is passed to a function or method by value
    - when an object is returned from a function or method by value.

13 Copy Assignment Operator

- Be able to write a copy assignment operator for a class that dynamically creates an array.

14 Abstract Data Type Definition

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

15 Stack and Queue ADTs

- Know the behavior that the stack and queue ADTs produce ("last in, first out" vs. "first in, first out")
    - stack is last in first out (LIFO)
    - queue is first in first out (FIFO)
- Know the types of errors that can occur when using a stack or queue (underflow on pop(), top(), front(), back())
- Be able to add an item to 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 assignement operator, destructor, etc.