**9/2**

Vocab:

* Single inheritance: derived class has a single base class
* Multiple inheritance: derived class has more than one base class
* Base classes: classes from which you derive new classes
* Composition (aggregation): one or more members of a class objects of another class type; a “has-a” relationship
* Derived classes: new classes that are created from base classes
* Encapsulation: the ability to combine data, operations on that data, in a single unit
* Inheritance: the ability to create new classes from existing classes; an “is-a” relationship
* Polymorphism: the ability to use the same expression to denote from a single class
* Destructors: used to deallocate dynamic memory allocated by the objects of a class

#ifndef H\_name (H\_name=header name)

#define H\_name

{

(code here)

}

#endif

ios is the base class for all stream classes

Class B: memerAccessSoecfiier A

* if memberAccessspecifier is public:
  + Public members of A are public in B, and can be directly accessed in class B
  + Protected members of A are protected in B and can be directly accessed by member functions (and friend functions) of B
  + Private member functions (and friend functions) of B
* If memberAccessSpecifier is protected:
  + Public members of A are protected members of B and can be accessed by the member functions (and friend functions) of B
  + Protected members of A are protected members of B and can be accessed by the member functions (and friend functions) of B
  + Private members of A are hidden in B and can be accessed only through public or protected members of A
* If memberAccessSpecifier is private:
  + Public members of A are private members of B and can be accessed by member functions of B
  + Protected members of A are private members of B and can be accessed by member functions (and friend functions) of B
  + Private members of A are hidden in B and can be accessed only through public/protected members of A

**9/11**

Exam 1 Study Guide:

* Know the key terms
  + Algorithm
    - A step by step solution to a problem in a finite number of stems and time
  + Data type
    - The domain of all values and the operations that can be preformed
  + UML and the symbols
    - - +, -, #
      * + public
      * - private
      * # protected
  + What UML is used for and why
* Classes
  + Define
    - Members
    - Standard functions
      * Accessor
      * Mutator
      * Destructor
      * Constructor
    - Fixed num of members
    - Default
      * Constructor
      * Destructor
  + Structured dt?
    - Class is a structured data type
  + Static vs auto
    - Static: lives throughout the entire code
    - Auto: lives only within the block of code
  + Destructor rules
    - 1 destructor within a class
* Cont
  + The use program is also called:
    - the client program
  + The file where function definitions go:
    - implementation file
  + Write a .h file
  + Know about constructors
    - Multiple constructors
    - U get a default constructor
    - Default constructor has an empty parameters list
    - Once u create a constructor u no longer get the default constructor
  + Pass by:
    - Value: making a copy of whatever I’m passing in
    - Reference: the memory location of whatever I’m referencing without making a copy
  + Understand the importance of + - #
  + Know member access operator
    - .
  + Know scope resolution operator
    - Snake bite ::
  + Const
  + Choose a correct class definition from code
  + How do u deal with avoiding multiple inclusions in a header file
    - Include 3 lines
      * Ifndef
  + Page 745 746 740 741
  + Is-a:
    - inheritance
  + Has-a:
    - aggregation or composition
  + How + - # work with
* Four standard member function types
  + Mutator
  + Accessor
  + Destructor
  + Constructor
* By default, all members are
  + Private
* Write a .h file based off a UML

**9/16**

Pointer variable: content is a memory address

Dynamic variables: created during execution

Address of operator: (&) a unary operator that returns the address of its operand

Member access operator arrow: ->

Dynamic array: array created during program execution

Shallow copy: when two or more pointers of the same types point to the same memory

Deep copy: when the contents of the memory pointed to by a pointer are copied to the memory location of another pointer

* Int \*p, q;
  + only p is the pointer variable
  + q is the int
* C++ does not automatically initialize variables
* An array name is a constant pointer

**9/21**

Memory type:

* Fixed
  + At the top of the memory locations, holds your executing programs
* Heap
  + Holds dynamic entries, contagious usage not a requirement
* Run-time stack
  + Built bottom up with push, pops

|  |
| --- |
| Fixed |
|  |
| Run-time stack |

Linked list:

* At a minimum a node contains:
  + Data
  + Pointer to the next node

**9/23**

#include <iomanip>

Cout << fixed << showpoint << setprecision(2);

Member access operator arrow (->):

* accesses class node components via the pointer
  + each node is made with new and has no name of it’s own
* by default pointers point to NULL

Read pages 852-868

**9/30**

Programming Project Due 10/14/20.

Information is on blackboard.

Quick Quiz due before class Friday.

852-859

Chapter 17: stacks and queues.

**10/5**

Recursion: solving a problem by reducing it to smaller versions of itself.

Recursive definition:

Recursive algorithm: finds a solution

Direct recursive: a function that calls itself

Indirectly

Infinite recursion: every recursive call results in another recursive call.

Iterative control structure: uses a loop to repeat a set of statements.

**10/7**

Do design a recursive function, you must do the following

* Understand the problem requirements
* Determine the limiting conditions
* Identify the base cases and provide a direct solution to each base case
* Identify the general cases and provide a solution to each general case in terms of smaller versions of itself

Homework: problems 8, 10, 12, 14. Page 1064

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

https://www.toptal.com/developers/sorting-algorithms

ADT stack - push, pop, track top, is\_empty, is\_full

ADT queue - enqueue, dequeue, track front and back, is\_empty, is\_full

**10/12**

Exception handling

* Exception: undesirable event detectable during program execution

Assert function:

* Checks if an expression meets certain conditions
* If conditions are not met, it terminates the program
* #include <cassert>
* assert(Condition);
* Assert checks whether an expression meets a specified condition; terminates if not met

try/catch

* Try
  + Throw *Value;*
    - Kinda like a return value
* catch(identifier)
* Statements that may generate an exception are placed in a try block
  + If no exception is thrown in a try block
    - All catch blocks are ignored
  + For try/catch to work, the exception
* Try/catch block handles exceptions

Stack unwinding

https://www.youtube.com/results?search\_query=linked+list+implementation+of+a+stack+c%2B%2B

https://www.youtube.com/results?search\_query=linked+list+implementation+in+a+queue+c%2B%2B

https://www.youtube.com/results?search\_query=array+implementation+in+a+queue+c%2B%2B

https://www.youtube.com/results?search\_query=array+implementation+of+a+stack+c%2B%2B

**10/14**

Exam 2 prep:

* ADT and Memory
  + What is it - define
  + Explain ADT list
    - Grouping of data that are somewhat related that can be sorted or unsorted
  + Three scopes and where they are in the memory, when created, when die
    - Global - stored in the fixed
    - Local - stored in the runtime stack
    - Dynamic - stored in the heap
  + Static
    - Use with local or dynamic
    - Bypasses normal scope rules
    - Desperation thing
* Classes/inheritance
  + Public
    - Public members of A are public in B, and can be directly accessed in class B
    - Protected members of A are protected in B and can be directly accessed by member functions (and friend functions) of B
    - Private member functions (and friend functions) of B
  + Private
    - Public members of A are protected members of B and can be accessed by the member functions (and friend functions) of B
    - Protected members of A are protected members of B and can be accessed by the member functions (and friend functions) of B
    - Private members of A are hidden in B and can be accessed only through public or protected members of A
  + Protected
    - Public members of A are private members of B and can be accessed by member functions of B
    - Protected members of A are private members of B and can be accessed by member functions (and friend functions) of B
    - Private members of A are hidden in B and can be accessed only through public/protected members of A
* Preprocessor directive
  + Explain:
    - #infdef H\_test
    - #define H\_test
    - #endif
* Order of Precedence
  + List, in order
    - ::
    - &
    - ()
    - ->
    - .
    - \*
  + Which has a higher precedence - postfix or prefix operators?
    - Postfix
  + Page 821
* Parameters
  + Formal
  + Actual
  + Value
  + Reference
* Pointers
  + Declare
  + Use
  + Evaluate us of
  + Define
  + Shallow vs deep copy
    - Page 839
    - Shallow
      * Means you have 2 things pointing at the same thing
    - Deep copy
      * Separate
* Recursion
  + What is a recursive algorithm
    - A problems solved by reducing the problem into smaller and smaller till you get a solution
  + What is needed for a recursive algorithm to work
  + Write a recursive function to calculate x!
  + Evaluate code
  + Types of recursion
    - Name and explain
  + Advantages and disadvantages of recursion as an implementation of an algorithm
    - Advantage
      * Its really easy to solve the problem and implement it
    - Disadvantage
      * Its slower
  + If you recursive call fails to quit what will, most likely, be the outcome
    - It will crash because it keeps calling itself
      * Stack overflow
* Exception
  + Define exception
    - An unexpected event that you didn’t mean to happen
    - Protecting yourself from user error
  + Explain assert - how to use, what is required
    - #Include <cassert>
  + use assert to test for eof
  + Explain try/catch
    - Rules
      * What you put in the try
      * What you put in the catch
  + Evaluate try/catch
* Stacks and queues
  + Stack
    - What is it
      * A data structure that follows the rules of a stack
    - What ops are used, what others should be included
      * is empty
      * Is full
      * Push
      * Pop
    - What is tracked
      * Top
  + Queue
    - What is it
      * First in first out
    - What ops are used, what others should be included
      * Enqueue
      * Dequeue
    - What is tracked
      * Front
      * Back

**10/23**

ADT:

* List
  + Properties
    - A way to have a grouping of elements
      * Sorted
      * Unsorted
  + Operations
    - Add
    - Remove
    - Create
    - Delete
    - Find
    - isempty
    - isfull
* Stack
  + Properties
    - How the elements are added
      * FILO (Fist in last out)
  + Operations
    - Push
    - Pop
    - isempty
    - isfull
* Queue
  + Properties
    - FIFO (First in first out)
  + Operations
    - Enqueue
    - Dequeue
    - Isempty
    - Isfull

Data structures:

* An implementation of an ADT
* Focus on storage and retrieval properties
* Based on:
  + Needs for usage of data
  + Data storage
  + Access
* Linked list
  + Single
  + Double
  + Circular
  + As implemented with new/delete
* Array
  + Indexed
  + Fixed size — can be dynamic or size assigned at run time
* Searching
  + The process of determining if the key (target) value is present in the data
  + Requires a method to do comparisons
  + Based on data structure and implementation
* Linear (sequential)
  + Assumes what?
    - Data is not sorted
  + Worst case
    - — not found = N comparisons
  + Best case
    - — first element = 1 comparison
  + Average case
    - — book (N+1)/2
* Binary
  + Assumes what?
    - Sorted list
  + Note book considers each comparison as a count of 2
  + Worse case
    - — not found = log(base 2)N+1
  + Best case
    - — first element = 1 comparison
  + Average
    - — normally not discussed — why?
      * Won’t find it
* Which list implementation
  + Do you sort?
  + Linked list or array?
  + When do you sort?
  + How do you sort?

**11/4**

Trees-ADT

* Root
  + Children
* If children don’t have any kids, it’s called a leaf
  + To reword, if a node has no children it’s called a leaf
* Binary tree
  + 2 children
* Binary search tree
  + First value is assigned a node
  + If next value is less then node
    - Its a left child
  + If next value is more then node
    - Its a right child
  + lchild-lltnk (left child-left link)
  + rchild-rlink (right child-right link)
  + Binary Search Tree is a node-based binary tree data structure
* Element of a tree is called a node
* In order traversal (LNR)
  + Traverse the left subtree
  + Visit the node
  + Traverse the right subtree
* Pre order traversal (NLR)
  + Visit the node
  + Traverse the left subtree
  + Traverse the right subtree
* Post order traversal (LRN)
  + Traverse the left subtree
  + Traverse the right subtree
  + Visit the node

**11/9**

* Trees - ADT
  + Think graphs
  + Trees have roots and branches
* Binary tree
  + Root
  + Each node can have at most 2 children
  + BST - the choice of where to place the incoming data is based on the first value read in which is stored in the root
* Binary search tree
  + Implementation of a binary tree following special rules
    - First value in gets stored in the node called root
    - Next value is compared to the value in root
      * If value is less than root the value is put in a node that is pointed to by the leftchild pointer in root
      * If value is greater than root the value is put in a node that is pointed to by the rightchild pointer in root
* Rules
  + From the definition of the node, it is clear that for each node:
    - The data is stored in info
    - A pointer to the left child is stored in lLink
    - A pointer to the right child is stored in rLink
* Terms
  + Tree
  + Root
  + Child
  + Leaf - no kids
  + Length - length of the path is number of branches on the path
  + Level - number of branches on the path from the root to node in question
  + Height - number of nodes on the longest path from root to leaf
  + Parent - node that has at least one child
  + Path - sequence of nodes to get from beginning to last node (you define)

**11/11**

Exam 3:

* Recursion
  + Understand how recursion works
  + How a recursive function works
  + Difference between base and general case
  + Indirect recursion
  + Be able to work through a recursive function and give the output given a call
* Linked list
  + What is it - define
  + What is a node
  + What is the special pointer
  + How can we build it
  + What types have we looked at
  + c++ operators
* Stacks and Queues
  + Define completely stacks
  + Define completely queues
* Searching
  + What types
    - Linear
    - Sequential
  + How it is done
  + Analysis
    - Binary is better
      * Trade off, must have a sorted list
* Sorting
  + Why?
    - to make it easier to find things
  + How specific algorithms are implemented
  + Be able to explain how different algorithms work
  + Analyze
* Trees
  + Compare and contrast BT and BST
    - BST compares values to get left and right
    - BST is a sorted tree
  + Know root, height, level, leaf, child, parent
    - Root -
    - Height - Number of nodes on the longest path from root to leaf
    - Level - Number of branches on the path from the root to node in question
    - Leaf - No kids
    - Child -
    - Parent - Node that has at least one child
  + Traversals
* Data types
  + Compare and contrast data types with abstract data type
    - the domain of all values and the operations that can be performed without regards to implementation

## 11/16

* Binding
  + compile-time binding
    - necessary code to call a specific function is generated by the compiler
    - static binding
    - early binding
  + Run-time binding
    - Binding happens at run-time
    - Compiler does not generate the code to call a function
    - Compiler generates enough to enable the run-time system to generate what is needed
    - Late binding
    - Dynamic binding
* Vectors and dynamic arrays
  + Define an array
  + Scope review
    - Fixed
    - Run-time stack
    - Heap
  + Run-time vs compile time
  + Any dynamic data type is created only when used during run-time
    - Heap
* Dynamic array
  + Dynamic array has a pointer to the array, rather than a set memloc
  + Memory not allocated until array is used
  + The existence if the data structure is dynamic
  + The base data structure remains the same
* Vectors
  + An array that can be resized during run time
  + Trade off is time to resize
* Typedef
  + Alias to a standard data type
  + From C is typedef char C;
    - So C is now a data type that references char
  + Using from c++
    - Using C = char;
    - Semantically equivalent
* Overloading
  + Two different functions can have the same name if at least one of their parameters is different
    - Different parm data type
    - Different number of parms
* Operator overloading
  + Having used before
    - - / % +
  + Allows programmer to exten the definitions of the operators
    - Relation
    - Arithmetic
    - Insertion
    - Extraction
  + Can overload
  + Cannot create new operators
  + Must write to define what operation is overloaded
    - What operation should be preformed
  + Syntax
    - returnType operator operatorSymbol(from parms)
    - cannot overload: . .\* :: ?: sizeof
  + Example:
    - Overload equality operator for class clockType:
      * Bool operator==(const clockType &otherClock) const

## 11/18

* Vector
  + Array that is dynamic
  + Allows for re-sizing during run-time
  + Not allocated at compile time
  + #include <vector>
  + Vectors are sequence containers representing arrays that can change in size
  + <http://www.cplusplus.com/reference/vector/vector/>
* Function templates
  + Write a single code segment for a set of related functions
  + Template <class Type>

declaration;

* + Type is data type
* Class templates
  + Write single code segment for a set of related classes
    - Template <class Type>

Class declaration

* + Parameterized types
  + Pg 969
  + 971 header file and imps
  + See SC ex 13-10 list type

## 12/2

Final exam review

* Chapter 10
  + Classes
    - Structured data type?
    - Reserved word?
    - Is an object of a class a class instance?
    - What is meant by instantiating a class?
      * Make instance of an object
  + What makes up a class?
    - Members
    - Member variables
    - Member functions
  + How does a class fit the definition of a data type?
  + What are the attributes that can be assigned to class members?
    - Private
    - Public
    - Protected
  + What is the default
    - Private
  + Be able to write a class definition from a UML diagram
  + What are the four categories of class member functions?
    - Accessor
    - Mutator
    - Constructor
    - Destructor
  + How many destructors can a class have?
    - 1
  + Two built in operators
* Chapter 11
  + list and explain the rule for inheritance and member access specifier
    - page 740, 763
  + be able to write code to inherit from a base class
    - write the derived class code
  + use appropriate code to prevent multiple inclusions of a header file when compiling
    - ifend (Look at notes to see spelling)
    - endif (look at notes to see spelling)
  + OOP and OOD
    - 3 basic principles and their definitions
      * Page 773
* Chapter 12
  + Pointers
    - What is it?
      * A memory location
    - Syntax?
    - Does the pointer data in c++ have a name associated with it? //
      * Simple
      * Structured
      * Pointer
    - \*
      * member access operator
    - &
      * Address of operator
    - ->
      * Member access arrow operator
    - To indicate that a pointer has no value in it you can assign:
      * 0 or NULL or NULL pointer
    - The only hardcoded number that can be assigned to a pointer is \_\_\_\_\_\_\_
      * 0
    - T/F you should use arithmetic operations on pointer values?
      * False
    - T/F you can compare pointer values
      * True
    - If a pointer is created during program execution it is a \_\_\_\_\_ value
      * Dynamic
    - The keyword \_\_\_\_\_\_ is used to create a dynamic variable
      * New
    - What is the difference between a dynamic array ad a local/global array?
      * Dynamic
        + Created at run time
        + Stored in the heap
    - Array Definition
      * A data structure that is a fixed size or a fixed number of elements all of the elements of the same data type that are referenced externally only by the address of the first element and then to get to the different elements you have to use indexing. What makes it different if it’s a dynamic array is it created at runtime stored in a heap and has a pointer to it that we actually consider to be a pointer.
      * Contiguous block of memory, comprised of homogeneous data types, elements accessed via indexes, data structure accessed via one memloc assigned to identified of the array (watch recording to place this in right question)
    - What is a shallow copy?
      * Second pointer that points to the same data
    - What is a deep copy?
      * Second block of data
* Chapter 13
  + Operator overloading
    - You can overload an operator
  + Operator that has different means with different data type is said to be overloaded
    - Give an example of a (standard) overloaded operator
  + Syntax
  + T/F operator functions in c++ are value-returning functions
  + 5 operators that can’t be overloaded
    - Member access operator
    - .splat
    - Scope resolution operator
    - Conditional operator
    - Size of
* Chapter 14
  + Exception handling
  + Why do you have to plan exceptions?
  + Define exception
  + What can you do with assert?
  + What needs to be included to us assert?
    - #include <cassert>
  + What is a try/catch block?
    - A way to protect against bad things from happening
  + Rules for how try/catch works
    - Different ways to catch it
* Chapter 15
  + Recursion
    - Define
      * Process of solving a problem by splitting it into smaller problems
    - Why is it important
      * Because it is a great way to solve problems
      * Easy way to solve problems
    - A general case must always resolve to \_\_\_\_\_\_\_\_
      * Base case
    - What is meant by direct recursion?
      * Calls itself
    - What is meant by indirect recursion?
      * Calls something else then calls back
    - Be able to write a simple recursive function
    - Be able to analyze a recursive function to determine output and/or errors
    - What can be a downfall of implementing a recursive function?
      * You can run out of memory
* Chapter 16
  + Linked list
    - Implements what ADT?
      * Lists
      * Stacks
      * queues
  + What you need to keep track of this with a linked list?
    - Head
  + A linked list is made up of \_\_\_\_\_\_\_\_\_\_. Which contain (at a minimum) \_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_
* Other
  + Binary tree
    - Number of branches in a path binary tree is length
    - Number nodes on the longest path of a binary tree is height
  + Know how to declare an array of pointers
    - Int \*array[max]
      * max is just a variable, put in number
  + Pass in a parameter by value
    - Make an entire copy of it
  + Pass by reference
    - Just send the memory location
  + Virtual and dynamic always bounded at run time (execution time)
  + If I have a data structure that allows me to make a complete copy from one to another, which operator would I use?
    - Done through the assignment statement
  + **Essay question:** Define a binary search tree
    - What it is
    - Anything special
      * The root
    - How it’s built
    - Advantages
    - How it could go wrong
      * N because you could end up with a completely squed tree
  + What the level of a root on a binary tree is
    - 0
  + Any recursive algorithm no matter how many base case it has the general solution must eventually resolve to \_\_\_\_\_\_
    - Base case
  + Function A calls function B, B calls C, C calls D, D calls A
    - Indirect recursion
  + Infinite recursive function
    - It’ll crash because it runs out of memory
  + let f be a function on n. by the term \_\_\_\_\_\_\_\_\_ we mean the study of the function f as n grows without bound
    - Asymptotic
  + In a bubble sort, if I have an array, the first step is to compare what?
    - First two terms, which would be indexed by 0 and 1