CS 32 Study Guide: Algorithms, Data Structure vcs, Abstract Data Types, Headers, Linked Lists, Stacks, Queues, Maps, Inheritance

An algorithm is a set of instructions/steps that solve a particular problem.
The imporance of algorithms is: RUNTIME

A data structure the data that's ope ated on by an algorithm to solve a problem.

Abstract Data Type (ADT): The collection of (a) data structures, (b) algorithms and (c) interface required to solve a particular problem. The ADT provides an interface to secret algorithms and data structures In C++, ADT's are defined as Classes

Object Oriented Programming: programs are co structed from multiple self-contained classes.

Examples of Algorithms:
- Linear search
- Binary search

```
/* NEVER INCLUDE A .CPP FILE
IN ANOTHER FILE. ONLY
INCLUDE .H FILES
NEVER PUT 'USING NAMESPACE
STD' IN A HEADER*/
```

Preprocessor Directives:

```
#ifdef FILE_H
 //checks if already defined
#ifndef FILE_H
 //checks if not defined
#define FILE_H
 //defines a constant
#endif //like an end bracket
```

01101010
01001100
01001011
10110101

```
/* use include guards
to prevent multiple
definitions */
```

constructors/destructors

```
/*if you declare an array of objects,
that object must have a default
constructor that requires no arguments*/
Class csNerd
{
  public:
    csNerd(int PCs, bool UsesMac)
      :m_numPCs(PCs), m_MacUser(UsesMac)
      //initializer list
    {…}
    ~csNerd(); //destructor, only one!
}
/*desctructors must: Free any dynamically
allocated memory, close any opened disk
files, and disconnect any opened network
connections*/
```

```
/* Class co position: If
a class contains one or
more classes as member
variables, */
```

```
/*include header files
when you define a
variable of that class
type or call any member
function from that
class.
DO NOT include header
files if you define a
parameter, return type
or pointer/reference
variable of the class */
class csNerd;//instead
```

Copying Stuff

```
Class Circ{
  public:
    Circ();
    Circ(const Circ& old);
      //copy constructor
    Circ& operator=(const Circ& source)
      //assignment operator
    {...
      return (*this); //required!
    }
}
int main(){
  circ one;
  circ two;
  two = one; //assignment operator call
  circ three(two); //copy constructor
}
```

```
/*a default copy constructor performs a shallow copy,
which does not work on dynamically allocated data or
opened system resources.
A copy constructor must:
-   determine how much memory is allocated by the old
    variable
-   allocate the same amount of memory in the new
    variable
-   copy the contents*/
```

```
/* the default assignment operator performs a shallow
copy, while will not work on dynamically allocated data
or any system resources that have been opened.
A assignment operator must:
-   free all dynamic memory used by the target instance
-   Re-allocate memory in the target instance to hold any
    member variables from the source instance
-   explicitly copy the contents of the source instance
    to the target instance*/
```

```
class Stack{
public:
  stack(); //constructor
  void push(int i); //add to stack
  int pop(); //remove from stack
  bool is_empty(void);
  int peek_top(); //return top value
  …
}
```

```
class Queue{
public:
  enqueue(int a); //adds a to end
  int dequeue(); //removes first
  bool isEmpty();
  int size();
  int getFront() //get front value
}
```

Linked Lists: (doubly linked)

```
struct node
{
  string name;
  node* next;
  node* prev;
}

class myLinkedList
{
public:
  void addtoFront(string name);
  void deleteItem(string name);
  void deleteItem(int slotNum);
  int find(string name);
  void print();
  myLinkedList() //creates empty list
  { first = last = NULL }
  ~myLinked List();
private:
  node* first //beg of list
  node* last  //end of list
```

```
/* You can create linked lists that are singly linked, doubly linked, or in a loop depending on what you need */
```

CHECK THE BOUNDARY CONDITIONS

Linked List Vs. Array
Array is Faster for
- getting a specific item
- less debugging problems

Linked List is Faster for
- inserting at the front removing from the middle

```
/*inert algrithms that insert at the top are the easiest to code and the fastest. Middle/end are slower/more complex*/
```

Circular Queue: use pointers head and tail to loop around an array

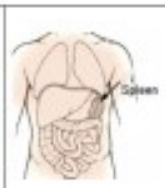
```
/* Destructors must traverse the entire linked list */
```

MAKE SURE THE POINTER DOESN'T POINT TO NULL

DESTRUCTING A DERIVED TYPE
1. Execute the body of the destructor
2. Destroy data members
3. Destroy base part

CONSTRUCTING A DERIVED TYPE
1. Construct base part
2. Construct data members
3. Execut the body of the constructor

```
/* Derived classes can only access public member variables and functions of the base class If you want Derived classes, but not the public to access variables, use protected*/
```




```
/* Copy Constructors and assignment operators will copy the base and derived data correctly, UNLESS it is dynamically allo aited */
```

RECURSION:

1. Identify if the problem is repetitive on a broad scale and/or can be simplified
2. Identify the simplist, complete case
3. Identify the base cases

```
if(base case)
    dosomething
else
    dosomething to reduce the size of the problem
```

Inheritance

```
class Base
{
  public
    Base(int p1, int p2)
    void doThis(); //!!!!!
    virtual void doIf(); //default: derived, if it exists
    virtual void doIf2() const =0; //pure virtual
  private:
    [stuff…]
}
class Derived : Public Base
{
  public
    Derived(int p1, int p2) : Base(p1, p2) {}
        //base must be constructed, or default is used
    virtual void doIf2() const;
        //declare overrides virtual as well
    virtual void doIf();
}

void Derived::doIf()
{    Base::doIf2();   }
//to call in a derived class a function from the base
//class that has been overwritten, you need to use
//'Base::'
```

```
/* Recursive functions should never use global, static, or member variables, only local variables and parameters! */
```

Generic Programming:
override/define generic comparison operators (<, >, ==, etc)
then, use templates! ☺