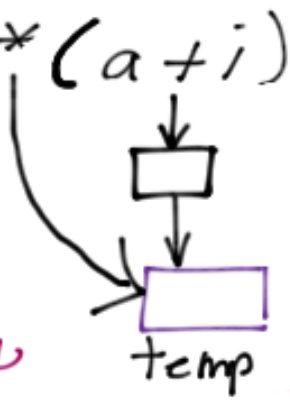
CS163 - Extra Slides

1. Reminder of Pointer Arithmetic and efficiency
2. Constructors with args
3. Function overloading

Pointer Arithmetic and Efficiency

1. Allows for quick access of memory when sequentially walking through an array or working with contiguous memory
2. As such, adding 1 to a pointer allows us to progress to the next element
3. Remember: `a[i] == *(a+i)`



Count the # of operations and fetches

4. Examine this code:

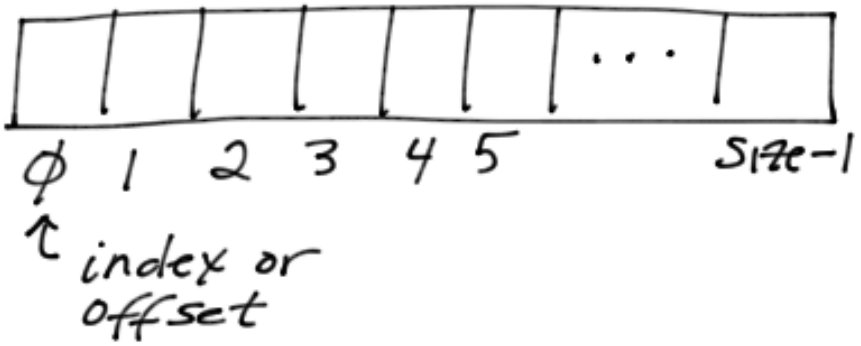
```
for (int i = 0; i<length; ++i)
    cout << array[i];
```

why not say: `i < strlen(array)`?

How many operations and fetches?

Pointer Arithmetic

char array[SIZE];



"address of" operator

```
char *ptr = array;
```

Data Type of element

```
ptr = &array[0];
      *array
      &(*array)
```

Now that a pointer is set up to point to the first element of a statically OR dynamically allocated array, we can use pointer arithmetic to access elements:

```
++ptr;
*ptr = 'a';
```

} OR `*(++ptr) = 'a';`

`cout << *(++ptr);`

what about `*ptr++ = 'a';`

Or, could we start elsewhere?

`ptr = &array[N];`

Some Class Construct Syntax:

1. constructors can have arguments:

```
class list
{ public:
    list();  ← default constructor
    list(int);  ← takes an integer arg
```

2. This is called "function overloading".
within the same scope you can create multiple functions with unique arguments and the right one will be invoked based upon the arguments in the function call.

3. Application or Client Program:

```
list mylist;  ← uses the default constructor
list another(10);  ← uses the one with an int
list many[100];  ← which one?
```

Interesting Syntax:

1. when creating an array of class objects, it is impossible to cause a constructor with an argument to be invoked
2. Therefore, ALWAYS provide a default constructor! `list();`
3. A default constructor is only automatically provided when NO constructors exist (and then it does nothing!)
4. This applies the same to statically and dynamically allocated arrays
5. Example: `list *ptr;`

   can point to one list object or the first of many

   `ptr = new list[size];`

   calls the default constructor size times!

One more example:

```
list *ptr = new list (num);
```

causes the constructor with an integer arg to be invoked

This creates only 1 object!
(not an array!)

Default Arguments

1. C# allows us to leave off arguments in a function call (working from the right to left) when default arguments are used.
2. Default Arguments are specified in the prototype, from right to left.

```
class list
{
  public:
    list(int size = SIZE);
```

const int SIZE = 100;

SIZE: a constant

Now, this one function is both the "default" constructor AND the constructor with an arg.

list obj; and list obj2(10);

Interesting facts about Default Arguments:

1. cin.get (3 argument) uses default arguments

for example:

istream & get(char *, int, char delimiter = '\n');

Default argument
so the right most arg
can be left off!

2. So, we use this function as:

cin.get(arrayname, size);

or cin.get(arrayname, size, '\t');

arrayname: or a pointer to a char!

'\t': new delimiter

(Remember, a pointer can point to one item or the first of many)

(Also - the name of an array IS a (constant) pointer to the 1st element

Example:

```
void function(int arg1, int arg2 = 10, int arg3 = 20);
```

With this example we can leave off the:

a) 2nd and 3rd args

b) Just the 3rd arg

Only Valid calls:

a) function (100, 200, 300);

b) function (100, 200);

c) function (100);

function();

(You can't leave off an argument in the middle!)

Implementing Slide #17 Constructor (Topic 2)

```
//Data members
private:
   data *d_array;
   int size_of_array;
   int num_of_items;
```

```
list::list(int size)
{
    if(size <= 0)
        size = SIZE;   //error correct
    d_array = new data[size];
    size_of_array = size;
    num_of_items = 0;
}
```

Summary of Data Structure choices:

1. Statically Allocated Arrays
   a. Requires the class designer to KNOW (guess) how many items there are.
   b. Every instance of the ADT will have the same size
2. Dynamically Allocated Arrays
   a. Waits until run time to allocate the array. Has an opportunity to get information from the application
   b. Each instance of the ADT can be sized appropriately.

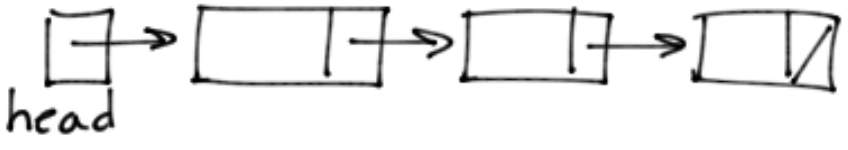
Other Data Structures

1. If the amount of memory needs cannot be known prior to creating the object (instance) a Linked Data structure may be best.

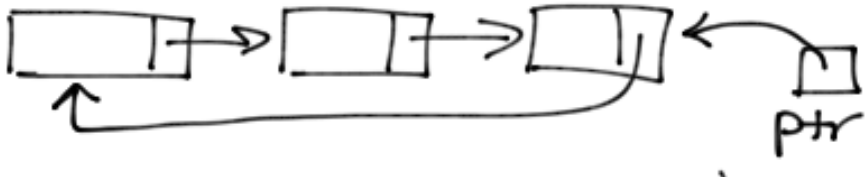
2. OR, if shifting (moving) data happens frequently

3. choices:

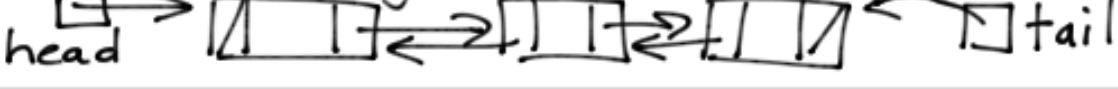
LLL (linear linked list)



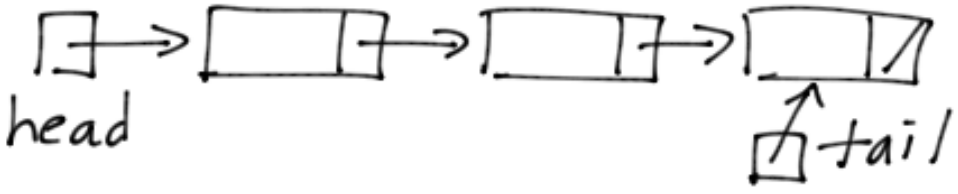
CLL (circular linked list)



DLL (doubly linked list)



With LLL we can also have a tail pointer:



This does increase overhead:

1. Adding:

```
if(!head)
{
    head = new node;
    tail = head;
```

2. Removing the only node becomes a special case:

```
if(head == tail)
{
    delete head;
    head = tail = NULL;
```

be careful!

why didn't we say: delete tail as well?

—SEG FAULT—

Never Return a struct or class object

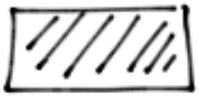
Use Arguments to "get back" values from a function.

Example of Poor Code:

```
data object;
object = listobject.retrieve(name);
```

if retrieve returns the found item

↓ Returns a Match



copy #1 into an un-named temporary on the program stack

The = operator makes copy #2

(If the object's class has dynamic memory and a destructor, these copies will cause "Shallow" copies of the pointers and then destruction of the original memory. THEN the same memory gets Released multiple times - SEG FAULT!

In CS202 we learn the solution:

a) Copy Constructor for the "returned" copy

b) = operator overloaded

CS163 Solution:

a) Never Return a struct or class object

b) Use arguments

c) Use pass by reference (where appropriate)