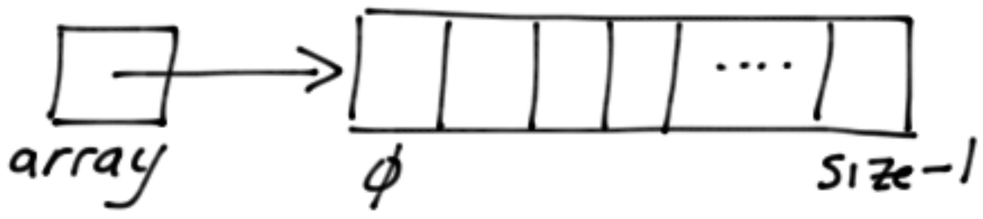
Today - Lecture 12 - CS162

1) Pointer Arithmetic
2) Introduction to "Linear Linked Lists"
3) Demonstrations

Dynamically Allocated Arrays

$$\texttt{char * array = new char[}\underbrace{\texttt{some\_size}}_{\text{desired size + 1}}\texttt{];}$$



Accessing the array can be done through the subscript operator:

$$\texttt{cout << }\underbrace{\texttt{array[i]}}_{\text{displays the character at index i}}\texttt{;}$$

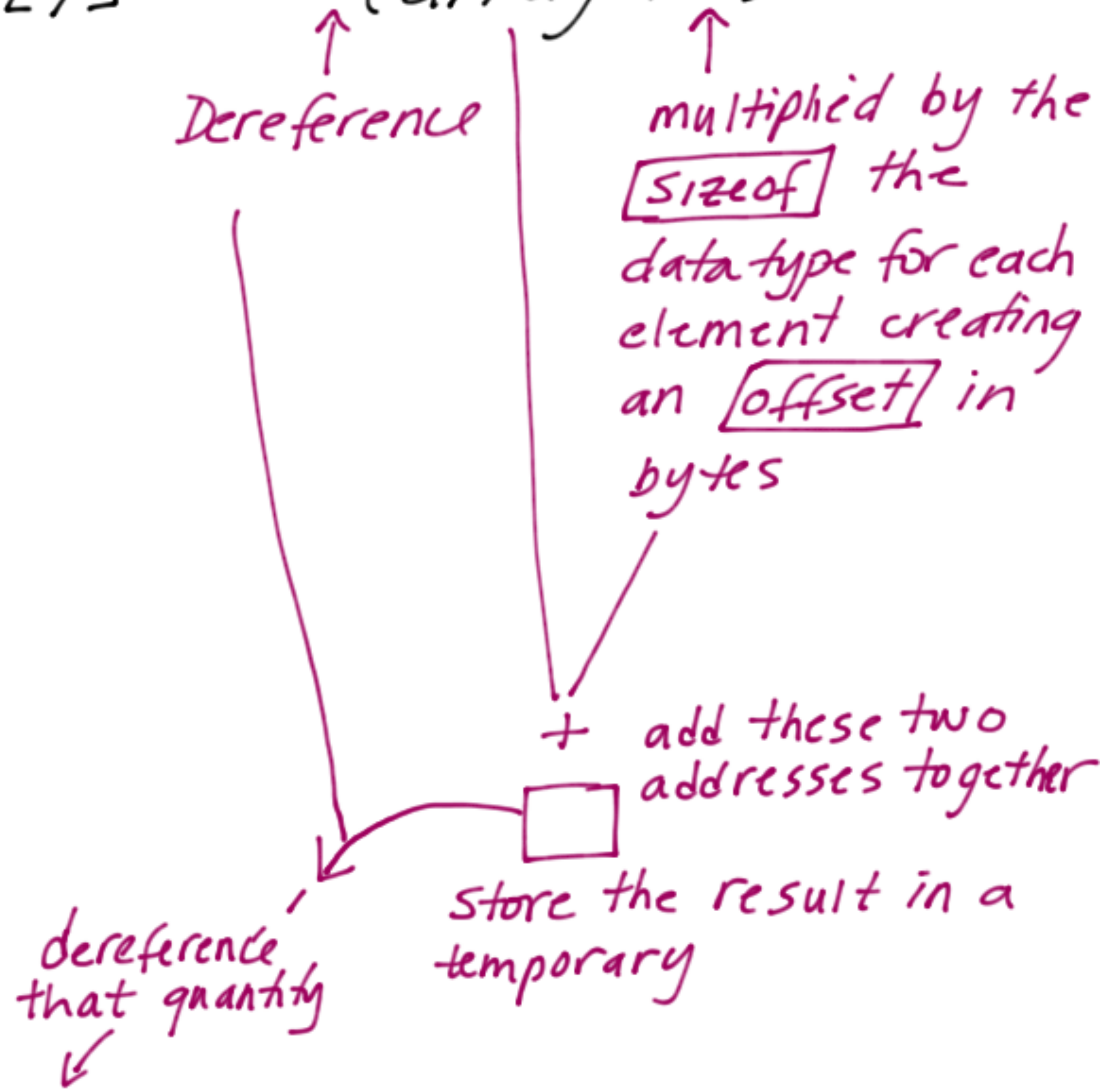
Or, use the cstring library (for arrays of characters):

```
length = strlen(array);
if (strcmp(array, "Karla") == ϕ)
```

Pointer Arithmetic

The subscript operator actually performs the following actions:

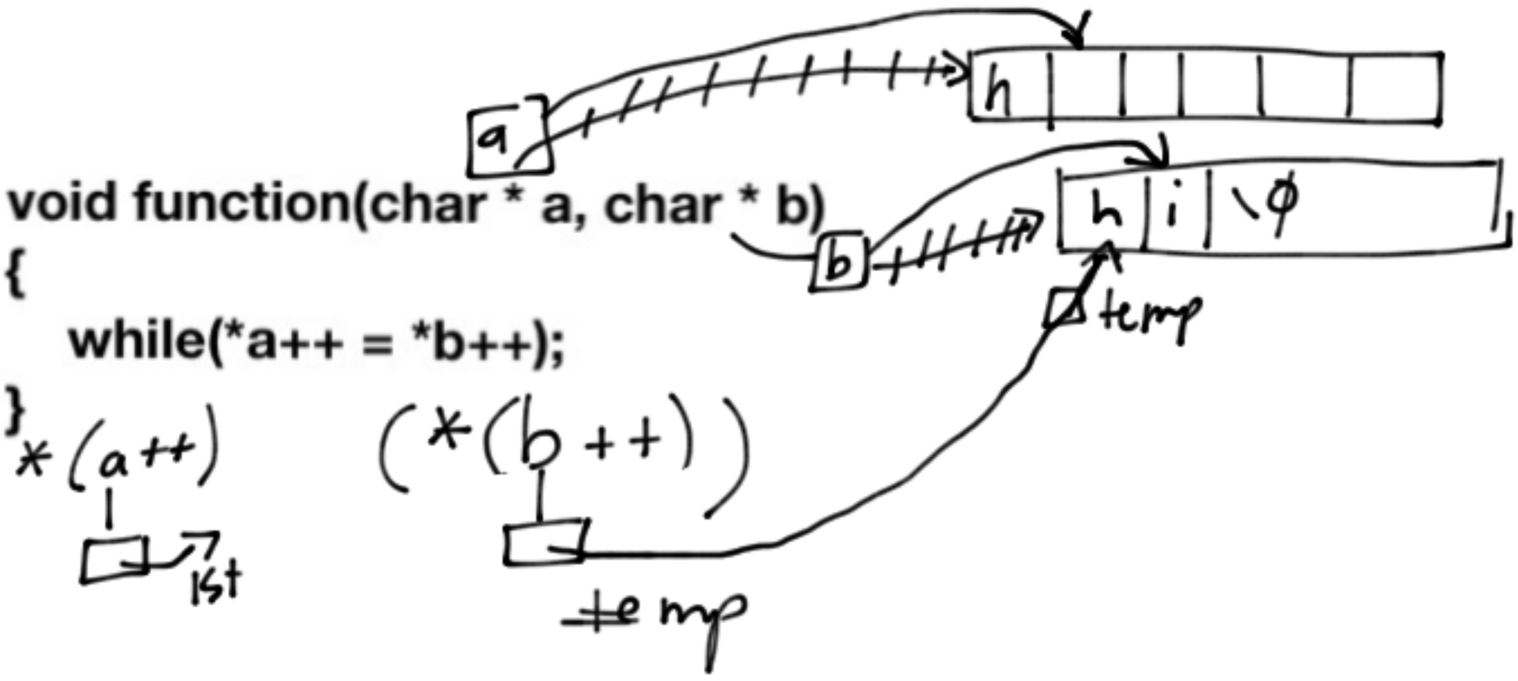
```
array[i] == *(array + i)
```



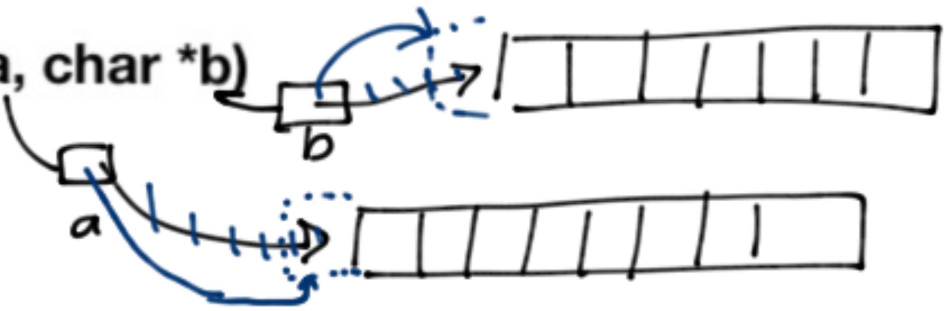
Dereference
multiplied by the $\boxed{\text{sizeof}}$ the data type for each element creating an $\boxed{\text{offset}}$ in bytes
+
add these two addresses together
store the result in a temporary
dereference that quantity

Look at this code:

```
void function(char * a, char * b)
{
    while(*a++ = *b++);
}
```

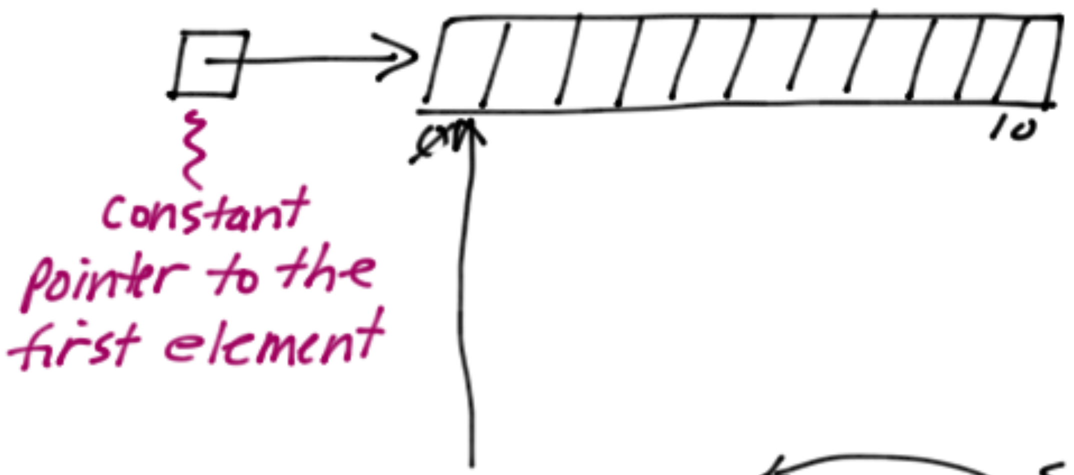


```
void function (char *a, char *b)
{
     --a; --b;

     while(*(++a)  = *(++b));

}
```



Pointer Arithmetic

char name [11];



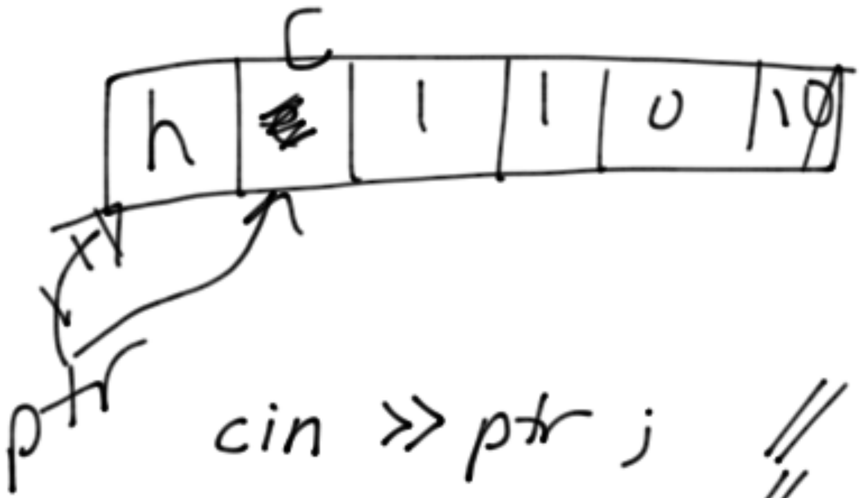
char *ptr = name; ← same as:
ptr = &name[0];

$\& \ *\underbrace{(name+0)}_{name}$

cancel out

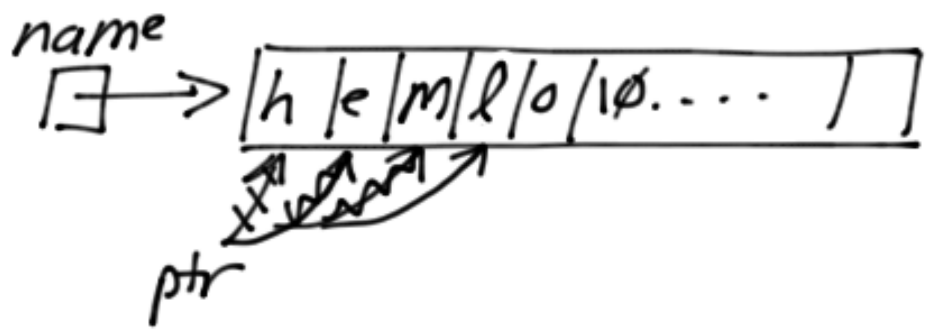
$\underbrace{}$

ptr = name



```
cin >> ptr;       // cin >> name;
++ptr;            // cout << name;
*ptr = 'c';
cout << *ptr;        cout << ptr;
```

Others:



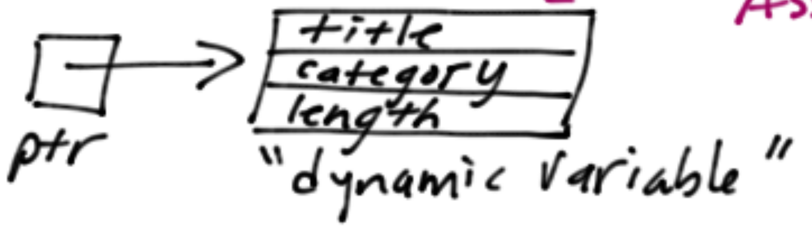
```
char *ptr = name;

cout << ptr;          // hello
cout << *ptr;         // h
cout << *(ptr++);     // h
cout << *(++ptr);     // l
cout << ++(*ptr);     // m
cout << ++ptr;        // lo
```

what is wrong with this:

```
*ptr++;        // Same ++ptr;

or

*(++ptr);      // same as ++ptr;
```

Pointers and Structures

```
video *ptr = new video;
```



Assume: char title[131];

```
cout << "Please enter the title";
cin.get(__________, 131);
```

what goes here?

ⓐ *ptr.title ← doesn't compile

ⓑ (*ptr).title ← compiles but.....

ⓒ ptr->title

called the indirect member access operator!

Pointer → member vs. object • member

Very Important

$$\underbrace{\text{object}}_{\text{struct or class}} \;.\; \text{member}$$

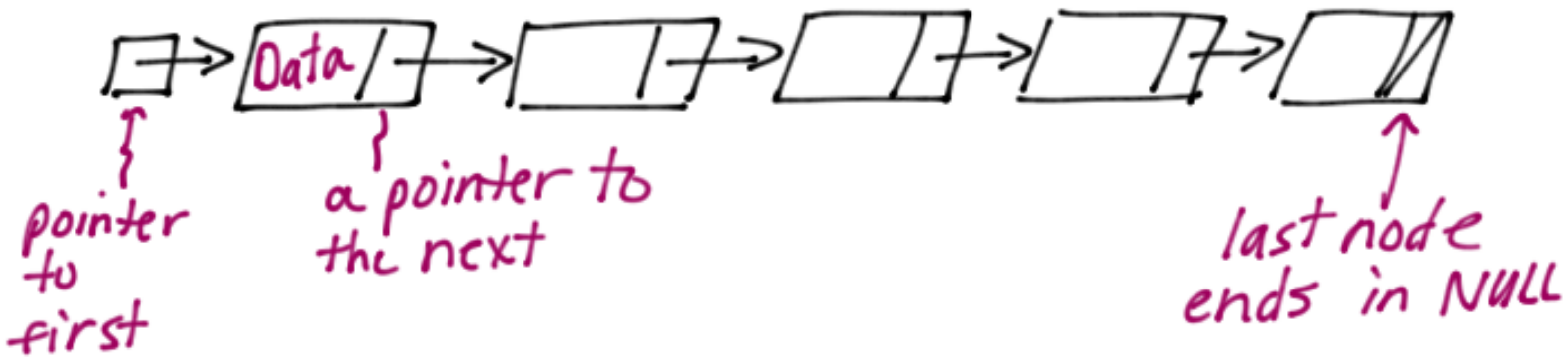
vs.

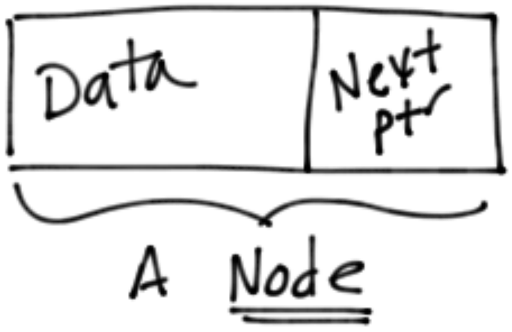
$$\underbrace{\text{Pointer}}_{\text{pointer to a struct or class}} \longrightarrow \text{member}$$

Make sure the pointer is $\boxed{\text{NOT}}$ NULL before dereferencing

Next topic: Linear Linked Lists

1) Flexible
2) Start with nothing & grow/shrink as needed



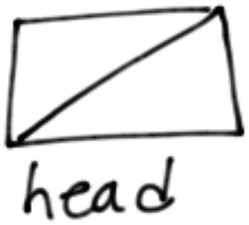


```
struct node
{
      video show;
      node * next;   ← called a recursive definition
};
```

Begins with ...

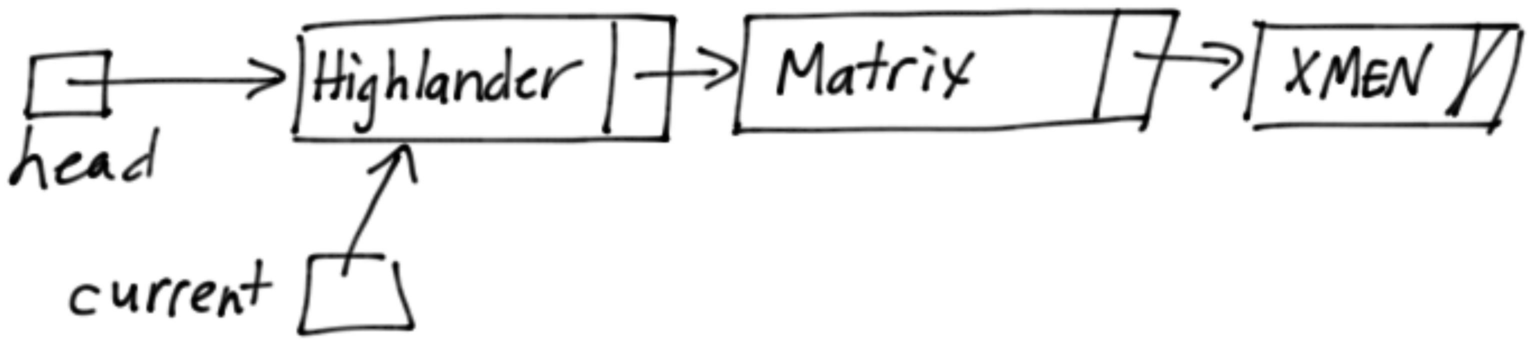
- "head" pointer
- which is a pointer to a node
- initialized to NULL for an empty list
- we can use other pointers to assist with traversal, creation, removal, retrieval

```
node * head = NULL;
```



represents an empty list. NO Items!

Examine this code to Traverse



```
node * current = head;
while (current != NULL)   //while (current)
{
    cout <<current->show.title <<endl;
    current = current->next;
}
```