Today - Lecture 11 - CS162

1) Introduction to Pointers
2) Dynamically Allocated Arrays
3) Using Dynamically Allocated Arrays in our show list program

- creating arrays sized just right at run time for names
- creating the array of items sized just right at run time
- deallocating (releasing) that memory

Announcements:

Pointers

1) A pointer "variable" holds a memory address

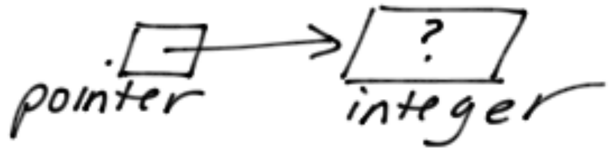
2) It is best to set pointers to NULL if they are not pointing anywhere

3) NULL is just a #define constant for Ø

```
int * pointer = NULL;
```

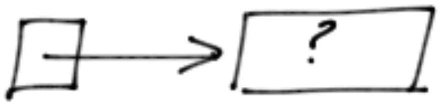
4) use "new" to allocate memory "dynamically" as the program is running

```
pointer = new int;
```



Dynamically Allocated Memory

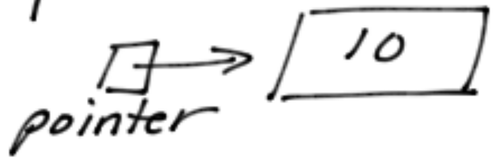
```
int *pointer = new int;
```



1) To Access The Memory we must get there <u>indirectly</u> through the pointer variable

This is called $\boxed{\text{DEREFERENCING}}$

```
*pointer = 10;
```



$$\texttt{cout << } \underbrace{\texttt{*pointer}}_{10}\texttt{;}$$

Deallocation of memory

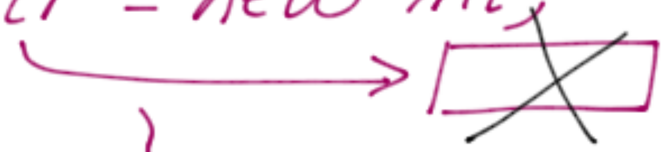
General

1) Memory for local variables (including pointer variables) is automatically released at the end of the block { } in which the variable was defined

2) $\boxed{\text{However}}$ memory allocated with $\boxed{\text{NEW}}$ is not automatically released until you use $\boxed{\text{delete}}$

3) For every use of new there should be a corresponding use of delete

```
int *ptr = new int;
```



```
delete ptr; // releases the memory
            // that ptr points to.
}
```

Dynamically Allocated Arrays

```
int length;
cout << "How many scores are there?";
cin >>length; cin.ignore(100,'\n');

float * scores;
float grade = 0.0;

scores = new float[length]; //dynamically allocated array

cout <<"Please enter " <<length <<" scores: ";
for (int i=0; i<length; ++i)
{
    cin >>scores[i];
    cin.ignore(100,'\n');
}

//calculate the average score
for (int i=0; i<length; ++i)
{
     grade+= scores[i];
}
grade /= length;

//we are done...
delete [] scores;
```

Pointer Arithmetic

$$array[i] == *(\underbrace{\underset{\text{Starting address}}{\underset{\uparrow}{array}} + \underset{\text{index is the offset}}{\underset{\uparrow}{i}}}_{\boxed{}\ \text{temp address}})$$

$$\downarrow \text{Go There!}$$

$$*(\underbrace{i + array})$$

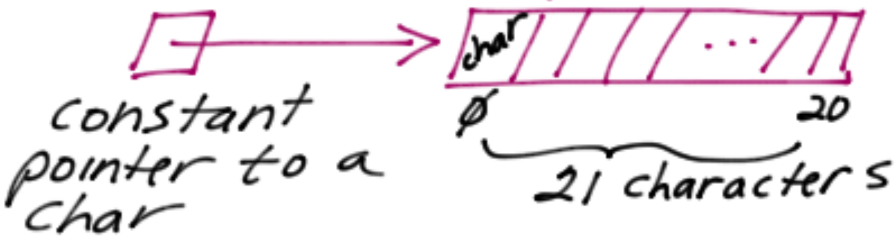
$$i[array]$$

Interesting...

a pointer can point to one item OR the first of many items contiguously arranged.

called a Statically Allocated Array

```
char name[21];
```

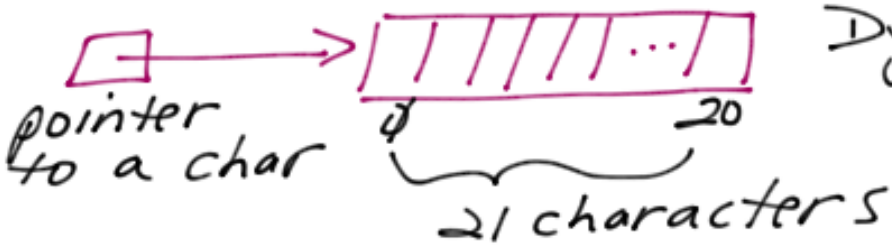


versus

this can be a variable!

```
char *name = new char[21];
```

Dynamically allocated array



```
delete [] name; //once we are done with
                // the array
```

Dynamically Allocated Array of characters

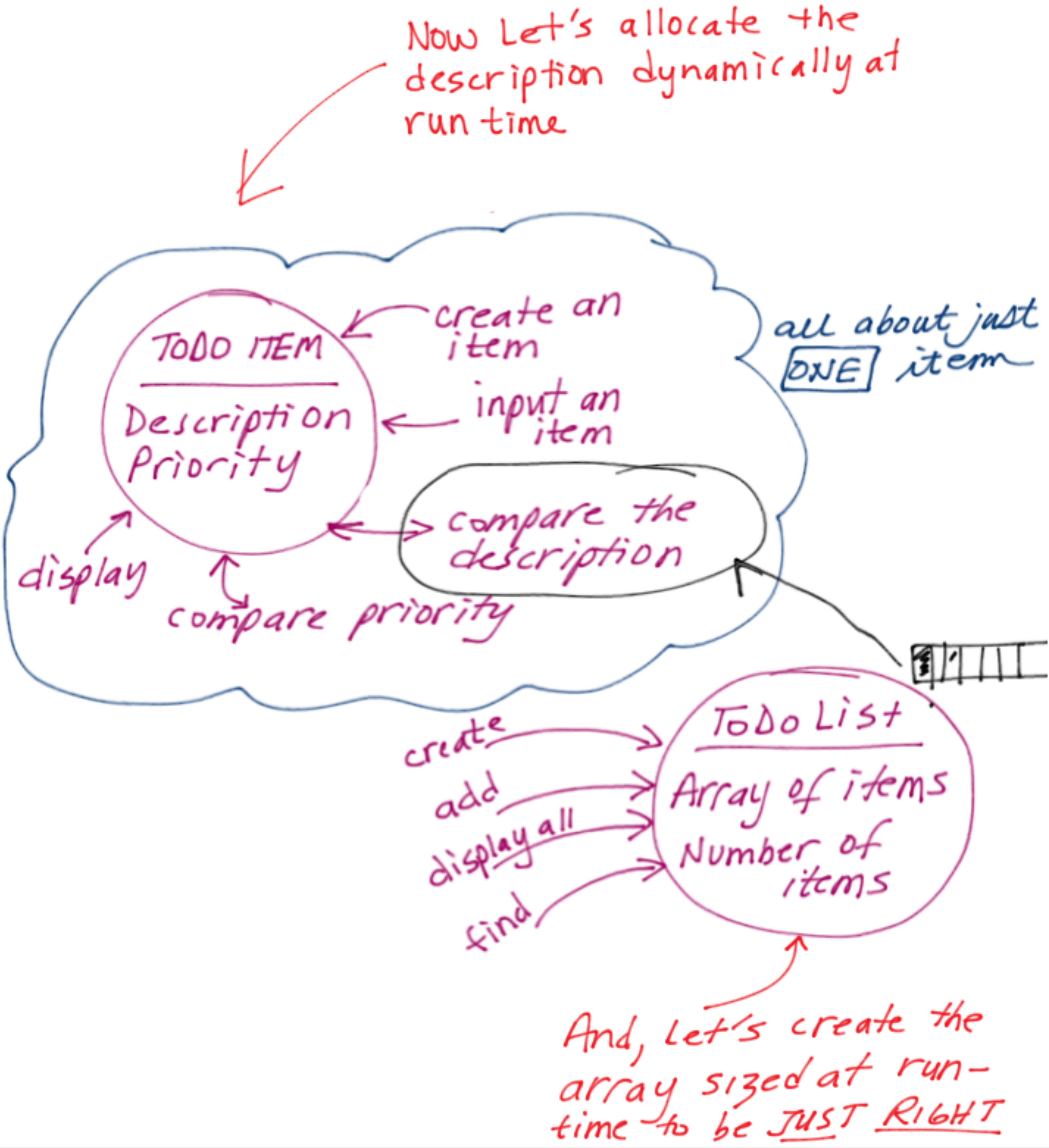
-sized just right!-

```
char temp[100];
char * name;

cout <<"Please enter your full name: ";
cin.get(temp,100,'\n');
cin.ignore(100,'\n');

name = new char[strlen(temp) + 1];
strcpy(name,temp);

cout <<"You entered " <<name <<endl;

//when done
delete [] name;
```

Examine These

(not all are correct)

```
char temp[100];
char * name;

cout <<"Please enter your full name: ";
cin.get(temp,100,'\n');
cin.ignore(100,'\n');

name = new char[strlen(temp) + 1];



name = new char[strlen(temp+1)];



name = new char[strlen(temp)] +1;
```


Now Let's allocate the description dynamically at run time
TODO ITEM
Description
Priority
create an item
input an item
compare the description
display
compare priority
all about just ONE item
ToDo List
Array of items
Number of items
create
add
display all
find
And, Let's create the array sized at run-time to be JUST RIGHT