Programming techniques

Week 3: Pointers and Dynamic Memory
Week 3

- Pointers and Dynamic Memory
  - What are pointers
  - Why dynamically allocate memory
  - How to dynamically allocate memory
  - What about deallocation?
  - Walk through pointer exercises
Pointers

- In C++, a pointer is just a different kind of variable.

- This type of variable points to another variable or object
  - (i.e., it is used to store the memory address of another variable nor an object).
  - Such pointers must first be defined and then initialized.
  - Then, they can be manipulated.
Pointers

- A pointer variable is simply a new type of variable.
  - Instead of holding an int, float, char, or some object's data,...it holds an address.
  - A pointer variable is assigned memory.
  - The contents of the memory location is some address of another “variable”.
  - Therefore, the value of a pointer is a memory location.
Pointers

- We can have pointers to (one or more)
  - integers
  - floating point types
  - characters
  - structures
  - objects of a class
- Each represents a different type of pointer
Pointers

- We define a pointer to an integer by:
  ```
  int * ptr; //same as int *ptr;
  ```
- Read this variable definition from right to left:
  - ptr is a pointer (that is what the * means) to an integer.
  - this means ptr can contain the address of some other integer
Pointers

- At this point, you may be wondering why pointers are necessary.
- They are essential for allowing us to use data structures that grow and shrink as the program is running.
- After midterm time we will learn how to do this...with linked lists
- We are no longer stuck with a fixed size array throughout the lifetime of our program.
Pointers

But first,

- we will learn that pointers can be used to allow us to set the size of an array at run-time versus fixing it at compilation time;
- if an object is a list of names...then the size of that list can be determined dynamically while the program is running.
- This cannot be accomplished in a user friendly way with simple arrays!
Defining Pointers

So, what are the data types for the following variables?

```c
int *ptr1, obj1;    //watch out!
char *ptr2, *ptr3;
float obj2, *ptr4;
```

What are their initial values (if local variables)?

--- yes, garbage ---
Defining Pointers

- The best initial value for a pointer is zero (address zero),
- also known as NULL (this is a #define constant in the iostream library for the value zero!)
- The following accomplish the same thing:

```c
int *ptr1 = NULL;
int *ptr2 = 0;
int *ptr3 (0);
```
Defining Pointers

- You can also initialize or assign the address of some other variable to a pointer, using the address-of operator.

```c
int variable;
int *ptr1 = &variable;
```
Allocating Memory

- Now the interesting stuff!

- You can allocate memory dynamically (as our programs are running) and assign the address of this memory to a pointer variable.

```c
int *ptr1 = new int;
```

```
ptr1  ?

dynamic variable
```
int *ptr1 = new int;

- The diagram used is called a pointer diagram
- It helps to visualize what memory we have allocated and what our pointers are referencing
- Notice that the dynamic memory allocated is of size int in this case
- And, its contents is uninitialized
- New is an operator and supplies back an address of the memory set allocated
Dereferencing

- Ok, so we have learned how to set up a pointer variable to point to another variable or to point to memory dynamically allocated.

- But, how do we access that memory to set or use its value?

- By **dereferencing** our pointer variable:

  ```
  *ptr1 = 10;
  ```
Dereferencing

- Now a complete sequence:

```cpp
int *ptr1;
ptr1 = new int;
*ptr1 = 10;
...
cout << *ptr1;  // displays 10
```
Deallocating

- Once done with dynamic memory, we must deallocate it
- C++ does not require systems to do “garbage collection” at the end of a program’s execution!

- We can do this using the delete operator:
  
  ```cpp
  delete ptr1;
  ```

  this does not delete the pointer variable!
Deallocation

- Again: this does not delete the pointer variable!
- Instead, it deallocates the memory referenced by this pointer variable
  - It is a no-op if the pointer variable is NULL
  - It does not reset the pointer variable
  - It does not change the contents of memory
  - *Let’s talk about the ramifications of this*...
Allocating Arrays

- But, you may be wondering: Why allocate an integer at run time (dynamically) rather than at compile time (statically)?

- The answer is that we have now learned the mechanics of how to allocate memory for a single integer.

- Now, let’s apply this to arrays!
Allocating Arrays

- By allocating arrays dynamically,
  - we can wait until run time to determine what size the array should be
  - the array is still “fixed size”...but at least we can wait until run time to fix that size
  - this means the size of a dynamically allocated array can be a variable!!
Allocating Arrays

First, let’s remember what an array is:

- The name of an array is a **constant address to the first element in the array**
- So, saying `char name[21];` means that name is a constant pointer who’s value is the address of the first character in a sequence of 21 characters
Allocating Arrays

- To dynamically allocate an array, we must define a pointer variable to contain an address of the element type.

- For an array of characters we need a pointer to a char:

  ```c
  char *char_ptr;
  ```

- For an array of integers we need a pointer to an int:

  ```c
  int *int_ptr;
  ```
Allocating Arrays

Next, we can allocate memory and examine the pointer diagram:

```c
int size = 21; // for example
char *char_ptr;
char_ptr = new char [size];
```

```
char_ptr
```

```
21 characters (elements 0-20)
```
Allocating Arrays

Some interest thoughts:

- the pointer diagram is identical to the pointer diagram for the statically allocated array discussed earlier!

- therefore, we can access the elements in the exact same way we do for any array:
  ```
  char_ptr[index] = 'a';  //or
  cin.get(char_ptr,21,'\n');
  ```
Allocating Arrays

- The only difference is when we are finally done with the array, we must deallocate the memory:

  ```cpp
  delete [] char_ptr;
  ```

- It is best, after doing this to say: `char_ptr = NULL;`
Allocating Arrays

- One of the common errors we get once allocating memory dynamically is a segmentation fault. It means you have accessed memory that is not yours.
  - you have dereferenced the null pointer,
  - you have stepped outside the array bounds,
  - or you are accessing memory that has already been deallocated.
In Review

- On the board, let’s walk through examples of the following:
  - allocating an array of integers dynamically
  - deallocating that array
  - writing a loop to set the values
  - now, allocate an array of video-structures dynamically
  - Show how you’d access the 3rd title
Pointer Arithmetic

- When we use the subscript operator, pointer arithmetic is really happening.
  - this means the following are equivalent:

    \[ \text{ptr1}[3] \quad == \quad *(\text{ptr1}+3) \]

- This means the subscript operator **adds** the value of the index to the starting address and then dereferences the quantity!!!
For Next Time

- Next time we will discuss:
  - more about pointers
  - integrating pointers and classes