CS 16 5/2

Pointers

(continues from lecture on 4/30)

Recall : pointers have the value of a memory address. The memory address that the pointer stores can have a value at it. This can be a little confusing. Remember that pointers hold memory addresses, not values at that address. You can access a value at a memory address by dereferencing a pointer.

Refer to code-from-class/05-02/pointers.cpp for the example given in class.

int\* p; // this is an example of a pointer to an int

int y;

p = &y; // this assigns the address of y to p.

// Remember p holds a memory address

\*p = 5; // this is an example of dereferencing a pointer. This accesses the value at the memory // address this changes the value of y

y++; // this changes the value of y from 5 to 6.

Warnings about pointers:

* Make sure you don’t dereference a pointer when you haven’t yet set to an address because you could end up with a segmentation fault (when you try to access memory you don’t have rights to).

Pointer to pointer

* Hold the address of another address that points to byte(s) of data.
* Next page(s) have an example of pointers to pointers

Sample file:

#include <iostream>

using namespace std;

int main()

{

 int \*\* A;

 int \* B;

 int \* C;

 int x, y;

 x = 3;

 cout << "x is " << x << endl;

 y = 247;

 cout << "y is " << y << endl;

 cout << "\n\n";

 B = &x;

 cout << "We set B = &x" << endl;

 cout << "A is at memory address (&A) == " << &A << endl;

 cout << "This is the same as the address of x (&x) == " << &x << endl;

 cout << "Notice that the derefereced B is the same value as x" <<

 " (\*B) == " << \*B << endl;

 cout << "\n\n";

 C = &y;

 cout << "We set C = &y" << endl;

 cout << "C is at memory address (&C) == " << &C << endl;

 cout << "This is the same as the address of y (&y) == " << &y << endl;

 cout << "Notice that the derefereced C is the same value as y" <<

 " (\*C) == " << \*C << endl;

 cout << "\n\n";

 A = &B;

 cout << "We set A = &B" << endl;

 cout << "A is at memory address (&A) == " << &A << endl;

 cout << "Notice that the derefereced A is at the same memory address as B" <<

 " (\*A) == " << \*A << endl;

 cout << "Now if we dereference A twice we should get the value of x (which is \*B) (\*\*A) == "

 << \*\*A << endl;

 cout << "\n\n";

 A = &C;

 cout << "One last time for clarity\n\nWe set A = &C" << endl;

 cout << "A is at memory address (&A) == " << &A << endl;

 cout << "Notice that the derefereced A is at the same memory address as C" <<

 " (\*A) == " << \*A << endl;

 cout << "Now if we dereference A twice we should get the value of y (which is \*C) (\*\*A) == "

 << \*\*A << endl;

 return 0;

}

Output:

x is 3

y is 247

We set B = &x

A is at memory address (&A) == 0x7ffee698d9d0

This is the same as the address of x (&x) == 0x7ffee698d9bc

Notice that the derefereced B is the same value as x (\*B) == 3

We set C = &y

C is at memory address (&C) == 0x7ffee698d9c0

This is the same as the address of y (&y) == 0x7ffee698d9b8

Notice that the derefereced C is the same value as y (\*C) == 247

We set A = &B

A is at memory address (&A) == 0x7ffee698d9d0

Notice that the derefereced A is at the same memory address as B (\*A) == 0x7ffee698d9bc

Now if we dereference A twice we should get the value of x (which is \*B) (\*\*A) == 3

One last time for clarity

We set A = &C

A is at memory address (&A) == 0x7ffee698d9d0

Notice that the derefereced A is at the same memory address as C (\*A) == 0x7ffee698d9b8

Now if we dereference A twice we should get the value of y (which is \*C) (\*\*A) == 247

Pointers and Arrays

Refer to pointers\_and\_arrays.cpp for examples.

We can use pointer arithmetic to move through arrays. If we set a pointer with the same type as an array (**int** myArr[]; **int** \*p;) we can set p = arr. This notation (p = arr) means p stores the memory address of the first element of the array in p.

Now \*p == myArr[0]; // is true

We can then use p = p + 1; // so now \*p == myArr[1] is true