05/07

Structs and References

References explained in code:

Sample file:

#include <iostream>

using namespace std;

int **main**()

{

 int num = 5;

 cout <<"First we do \n\nint num = 5;\nint & myRef = num;\n" << endl << endl;

 cout << "Num is the value (num) " << num << " and is at the address (&num) " << &num << endl;

 int & myRef = num;

 cout << "myRef has the value (myRef) " << myRef << " and is at address (&myRef) " << &myRef << endl;

 cout << endl << endl << "Notice that these are the same values and address which is not true if we do something like the following:" <<

 " \n\nint a = 5;\nint b = a;\n" << endl;

 int a = 5;

 cout << "a is the value (a) " << a << " and is at the address (&a) " << &a << endl;

 int b = a;

 cout << "b has the value (b) " << b << " and is at address (&b) " << &b << endl;

 cout << endl << endl << "Notice that these are the same values BUT different addresses." << endl;

 return 0;

}

Output:

First we do

int num = 5;

int & myRef = num;

Num is the value (num) 5 and is at the address (&num) 0x7ffee21df9f8

myRef has the value (myRef) 5 and is at address (&myRef) 0x7ffee21df9f8

Notice that these are the same values and address which is not true if we do something like the following:

int a = 5;

int b = a;

a is the value (a) 5 and is at the address (&a) 0x7ffee21df9ec

b has the value (b) 5 and is at address (&b) 0x7ffee21df9e8

Notice that these are the same values BUT different addresses.

Code from class has more examples.

Structs

Basically allow you to declare new data types which are composed of other data types (even other structs).

There are a couple of ways to declare instances of structs.

We could declare two instances of type box (a,b) like the sample file and output below:

Sample File:

#include <iostream>

#include <string>

using namespace std;

struct box{

 int width;

 int height;

 string name;

} a, b;

int **main**()

{

 cout << "Now we have two new instances of box!" << endl;

 a.width = 2;

 a.height = 5;

 a.name = "Box1";

 b.name = "Box2";

 b.width = 1234;

 b.height = 123;

 cout << "Box a has attributes \n\nname: " << a.name << "\nheight: " << a.height << "\nwidth: " << a.width << endl << endl;

 cout << "Box b has attributes \n\nname: " << b.name << "\nheight: " << b.height << "\nwidth: " << b.width << endl << endl;

 return 0;

}

Output:

Now we have two new instances of box!

Box a has attributes

name: Box1

height: 5

width: 2

Box b has attributes

name: Box2

height: 123

width: 1234

**OR we can do something like this:**

#include <iostream>

#include <string>

using namespace std;

struct box{

 int width;

 int height;

 string name;

};

int **main**()

{

 box a = {2,5,"Box1"}, b;

 cout << "Now we have two new instances of box!" << endl;

 b.name = "Box2";

 b.width = 1234;

 b.height = 123;

 cout << "Box a has attributes \n\nname: " << a.name << "\nheight: " << a.height << "\nwidth: " << a.width << endl << endl;

 cout << "Box b has attributes \n\nname: " << b.name << "\nheight: " << b.height << "\nwidth: " << b.width << endl << endl;

 return 0;

}

Which has the same output:

Now we have two new instances of box!

Box a has attributes

name: Box1

height: 5

width: 2

Box b has attributes

name: Box2

height: 123

width: 1234

Example with struct inside a struct:

#include <iostream>

#include <string>

using namespace std;

struct box{

 int width;

 int height;

 string name;

};

struct jackInTheBox{

 box jack;

 int secondsUntilPop;

};

int **main**()

{

 jackInTheBox a = {{2,5,"Box1"},500}, b;

 b.jack.name = "Box2";

 b.jack.width = 1234;

 b.jack.height = 123;

 b.secondsUntilPop = 501;

 cout << "jackInTheBox a has attributes \n\njack.name: " << a.jack.name << "\njack.height: " << a.jack.height << "\njack.width: " << a.jack.width << endl << endl;

 cout << "jackInTheBox b has attributes \n\njack.name: " << b.jack.name << "\njack.height: " << b.jack.height << "\njack.width: " << b.jack.width << endl << endl;

 cout << "Additionally jackInTheBox a has " << a.secondsUntilPop << " seconds until pop." << "\n\n";

 cout << "And jackInTheBox b has " << b.secondsUntilPop << " seconds until pop." << "\n\n";

 return 0;

}

Output:

jackInTheBox a has attributes

jack.name: Box1

jack.height: 5

jack.width: 2

jackInTheBox b has attributes

jack.name: Box2

jack.height: 123

jack.width: 1234

Additionally jackInTheBox a has 500 seconds until pop.

And jackInTheBox b has 501 seconds until pop.

Pointer Arithmetic

int \*ptr, ptr1, ptr2 = // some valid address

**VALID**

ptr + 1 // does the same as the next line

1 + ptr // move to the next piece of memory

ptr – 1 // move to the previous piece of memory

ptr1 – ptr2 // find how many spaces these two ptrs are away from each other

ptr == ptr // if true these pointers at the same address

ptr == 0 // if true this pointer has not been assigned yet

ptr == NULL // this is equivalent to ptr == 0

**INVALID**

ptr + ptr // could be out of accessible memory

1 – ptr

1 == ptr

LLDB and GDB

To use: compile your program with the -g flag ( e.g. g++ -g FILENAME.cpp -o FILENAME ).

Then run with lldb FILENAME. Or gdb FILENAME

Use (run) to start the program (don’t put it in parentheses).

Use (print NAME\_OF\_VARIABLE) to print out values of local variables.