Variables and Memory; Pointers

ITP 165 – Fall 2015

Week 10, Lecture 2

Memory

- Random access memory (***RAM***) is where data needs to actively be in order to operate on

- Even when reading input from a file – the file (or part of it) is loaded into memory behind the scenes

- Remember that 32-bit computers are limited to about 4 GB (~4 billion bytes) of RAM

Variables and Memory

- When we declare variables, those variables are created in memory

- For example:

```
int main() {
    int x = 0;
    double y = 5.0;
    return 0;
}
```

- Both x and y are variables, so they are created in memory

Variables and Memory, Cont'd

- The amount of memory a variable takes up depends on the type:

Type	Size
`int`	4 bytes
`char`	1 byte
`float`	4 bytes
`double`	8 bytes
`short`	2 bytes
`long long`	8 bytes

Variables and Memory, Cont'd

- The amount of memory an array takes up is the number of elements multiplied by the size of each element

- Some examples:

```
int test1[10]; // 4 * 10 = 40 bytes

char test2[20]; // 1 * 20 = 20 bytes

double test3[10]; // 8 * 10 = 80 bytes
```

Memory Address

- When we put something in memory, it has to go to a specific location or ***memory address***

- These addresses are usually written in hexadecimal, and look something like: `0x001BF92C`

- For simplicity, I will only use memory addresses up to and including `0xFF` (or 255 in decimal) for the following slides

Declaring Variables, Step by Step

```
int main() {
    int x = 0;
    double y = 5.0;
    return 0;
}
```

	Memory	
Variable	Address	Value

Declaring Variables, Step by Step

```
int main() {
    int x = 0;
    double y = 5.0;
    return 0;
}
```

	Memory	
Variable	Address	Value
x	0x04	0

Declaring Variables, Step by Step

```
int main() {
    int x = 0;
    double y = 5.0;
    return 0;
}
```

	Memory	
Variable	Address	Value
x	0x04	0
y	0x08	5.0

Declaring Variables, Step by Step

```
int main() {
    int x = 0;
    double y = 5.0;
    return 0;
}
```

	Memory	
Variable	Address	Value

Another Memory Example

```
int Array[] = {
    5,
    10,
    15,
    20,
    25
};
int x = 7;
```

	Memory	
Variable	**Address**	**Value**
Array[0]	0x04	5
Array[1]	0x08	10
Array[2]	0x0C	15
Array[3]	0x10	20
Array[4]	0x14	25
x	0x18	7

Address-of Operator

- So far, we have used the ampersand for several things:
  - && for Logical AND
  - & for declaring a parameter to be passed by reference

- But there's yet another use of the ampersand!

- Given a variable that was declared on a previous line, an & in front of the variable gets the memory address

Address-of Example

```
int x = 0;
double y = 5.0;

// Would output 0x08
std::cout << &y;
```

	Memory	
Variable	Address	Value
x	0x04	0
y	0x08	5.0

Address-of (Full Example)

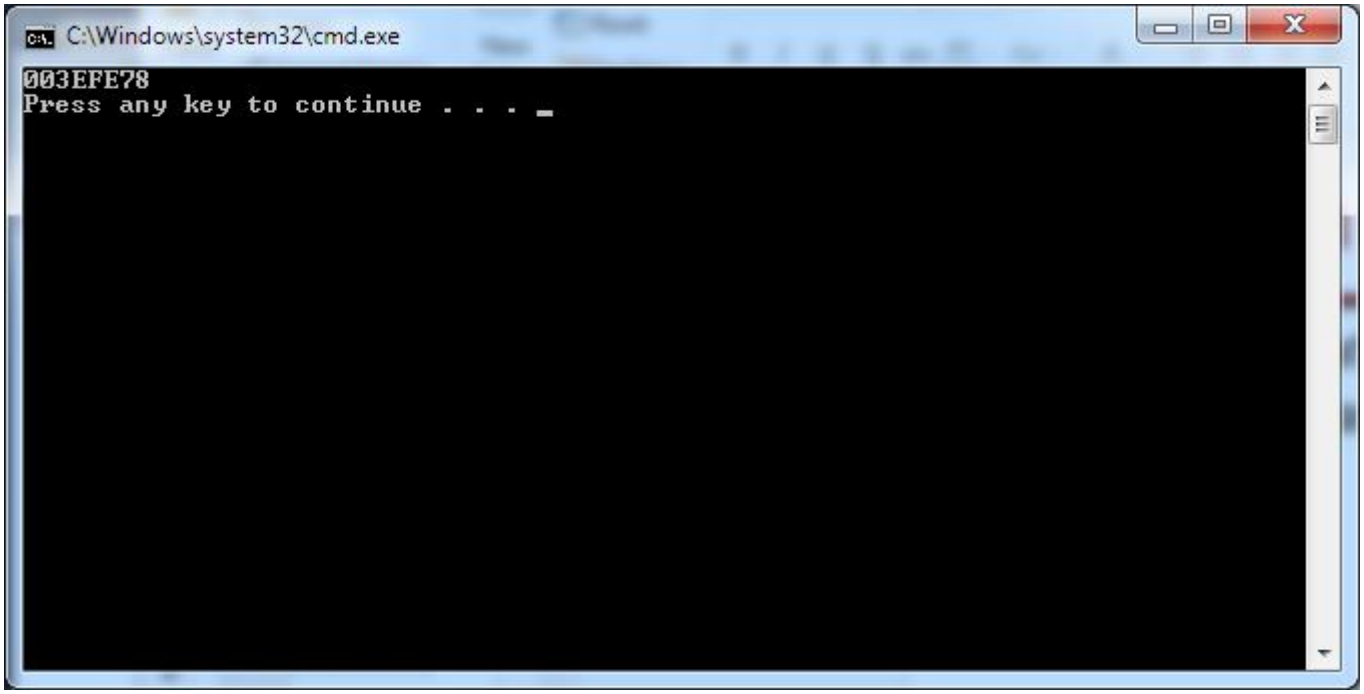
```
#include <iostream>

int main() {
    int x = 0;
    double y = 5.0;

    std::cout << &y << std::endl;

    return 0;
}
```

Address-of Full Example



- This means that this time we ran the program, the address of y was `0x003EFE78`

Another Address of Example

```
int Array[] = {
    5,
    10,
    15,
    20,
    25
};
int x = 7;
// &x == ??
// &(Array[0]) == ??
// &(Array[4]) == ??
```

	Memory	
Variable	Address	Value
Array[0]	0x04	5
Array[1]	0x08	10
Array[2]	0x0C	15
Array[3]	0x10	20
Array[4]	0x14	25
x	0x18	7

Another Address of Example

```
int Array[] = {
    5,
    10,
    15,
    20,
    25
};
int x = 7;
// &x == 0x18
// &(Array[0]) == 0x04
// &(Array[4]) == 0x14
```

	Memory	
Variable	Address	Value
Array[0]	0x04	5
Array[1]	0x08	10
Array[2]	0x0C	15
Array[3]	0x10	20
Array[4]	0x14	25
x	0x18	7

Another Address of Example

- When calling the addresses of an array, we can refer to the first element in three different ways:

1. &Array[0]
2. &Array
3. Array

- These are all equivalent, the address-of operator (without any brackets) will always reference the FIRST element in the array (#3 is something we'll cover soon)

Address-of Operator

- We can call adjacent memory slots using the “address-of” operator

- For example:

```
int Array[] = {
    5,
    10,
    15,
    20,
    25
};
```

Address-of Operator

```
int Array[] = {
    5,
    10,
    15,
    20,
    25
};

//Produces the memory address at index 0
std::cout << &Array << std::endl;
```

Address-of Operator

```
int x[] = {5, 10, 15, 20, 25};
for (int index = 0; index < 5; index++)
{
    std::cout << x[index] << ": " << &x[index] << std::endl;
}
```

Address-of Operator

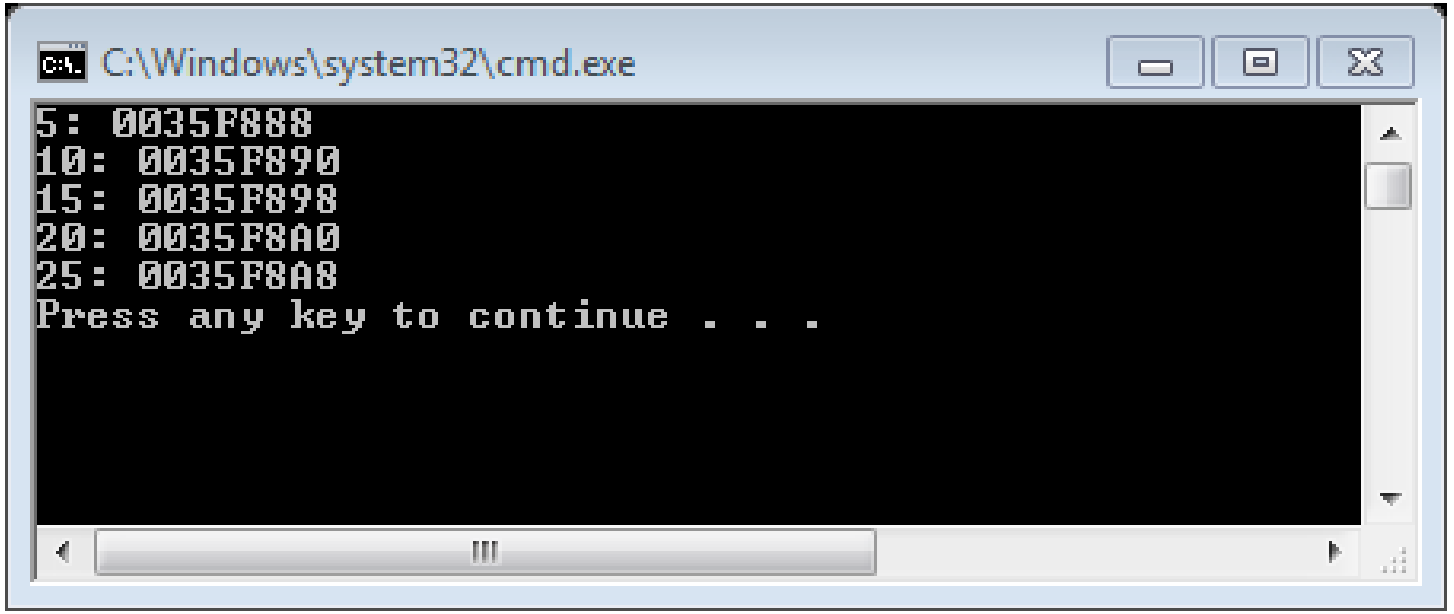
```
int x[] = {5, 10, 15, 20, 25};
for (int index = 0; index < 5; index++)
{
    std::cout << x[index] << ": " << &x[index] << std::endl;
}
```

Address-of Operator

```
double x[] = {5, 10, 15, 20, 25};
for (int index = 0; index < 5; index++)
{
    std::cout << x[index] << ": " << &x[index] << std::endl;
}
```

Address-of Operator

```
double x[] = {5, 10, 15, 20, 25};
for (int index = 0; index < 5; index++)
{
    std::cout << x[index] << ": " << &x[index] << std::endl;
}
```



Pointer

- A ***pointer*** is a type of variable that stores a memory address

- Since memory addresses are hexadecimal numbers, from the perspective of the computer there really isn't much of a difference between an integer and a pointer

- However, when we use a pointer, we are telling C++ that we're using this number as a memory address

Declaring a Pointer

- You have to declare a pointer just like any other variable

- To signify it's a pointer, you put an * between the type and the variable name

- Example:

```
int x = 0;
// Declare z as a pointer to an integer.
// Initialize its value to the address of x.
int* z = &x;
```

Declaring a Pointer Example

```
int x = 0;

int* z = &x;
```

	Memory	
Variable	Address	Value
x	0x04	0
z	0x08	??

Declaring a Pointer Example

```
int x = 0;

int* z = &x;
```

	Memory	
Variable	Address	Value
x	0x04	0
z	0x08	0x04

Null Pointer

- A ***null pointer*** is a pointer that currently does not have a valid memory address

```
// Initialize z to a null pointer
int* z = nullptr;
```

- When you declare a pointer, you should always initialize it to the address of a variable, or to a null pointer

- Internally, a null pointer is just a 0. So you could also do:

```
int* z = 0;
```

Declaring a Null Pointer Example

```
int x = 0;
double y = 5.0;

int* z = nullptr;
```

	Memory	
Variable	Address	Value
x	0x04	0
y	0x08	5.0
z	0x10	0x0

Another Pointer Example

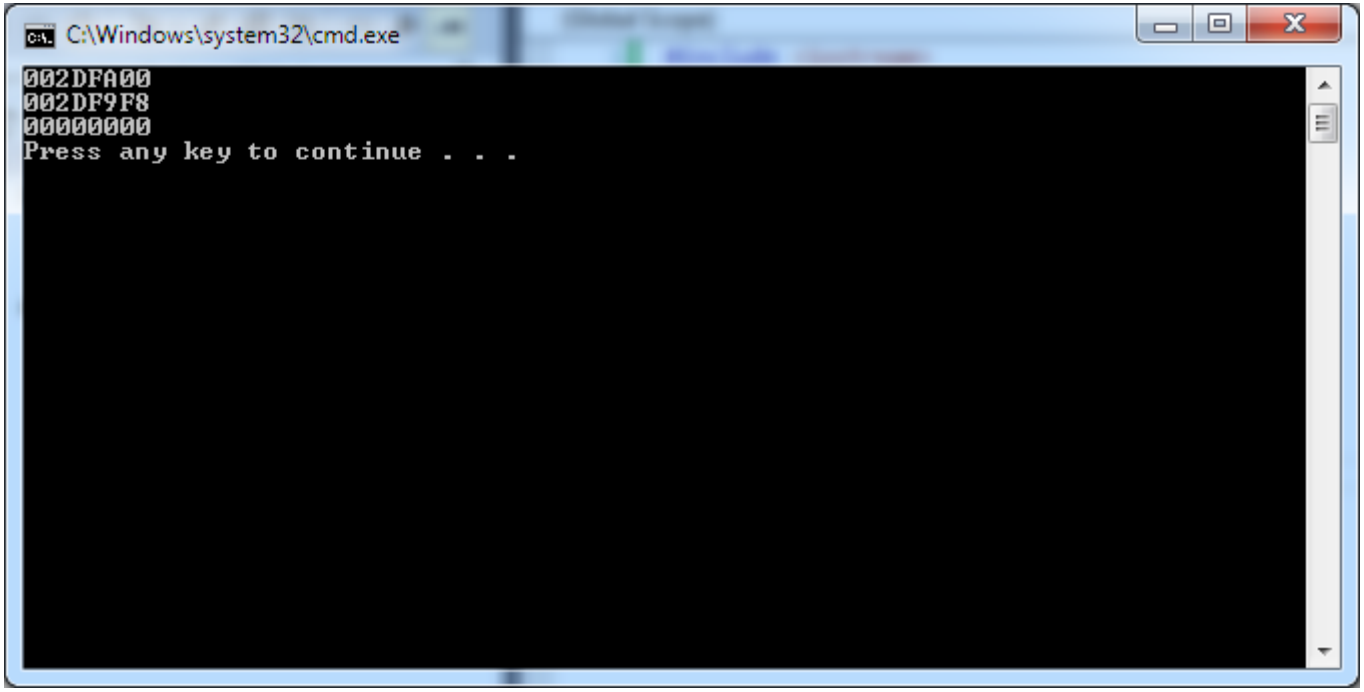
```
#include <iostream>

int main() {
    int x = 0;
    double y = 5.0;
    // Declare some pointers
    int* p1 = &x;
    double* p2 = &y;
    float* p3 = nullptr;

    std::cout << p1 << std::endl;
    std::cout << p2 << std::endl;
    std::cout << p3 << std::endl;

    return 0;
}
```

Another Pointer Example



Lab Practical #17