CSCI 2270: Data Structures

Recitation #2 (Section 101)

Introduction and Office Hours

- Name: Himanshu Gupta
  Email: himanshu.gupta@colorado.edu

- **Office Hours - 10am to 2pm on Mondays in ECAE 128**
  - In case that doesn't work for you, shoot me an email. We will figure something out that works for both of us.
  - Also, you can attend any TA's office hours. Timings are available on moodle in calendar.

- I am pretty approachable. So, ask as many questions as you can (both during the recitation and my office hours or via email). I am not your instructor, I am your TA.

Today's Agenda

- Review (20 ~ 30 mins)
  - Pointers and pointers to struct
  - Call-by-Value
  - Call-by-Reference
  - Call-by-Pointers

- An informal Quiz

- Exercise
  - Swap elements of an array using pointers
  - Reverse an array using pointers

Takeaways from Recitation 1 & Assignment 1

- Have finished VS Code setup and installed G++
- How to run your code?
  - Two step process:
    - Compile your code: **g++ -std=c++11 filename.cpp -o executable_name**
    - Run your executable file: **./executable_name** (Default name is **a.out**)
- Assignment 1
  - Read the error logs carefully
    - Practice Googling. It is an art you master with time and practice.
    - But don't Google for hours. If stuck, **ASK FOR HELP!**
  - Local environment and CodeRunner is slightly different
    - e.g., command-line arguments in different order

Pointers

WHAT ARE THEY???

Pointers

Pointers (after some time)

Computer Memory

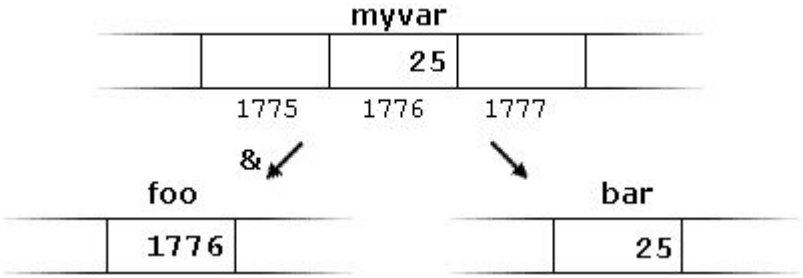
- Each variable you create is assigned a location in the computer's memory!
- A computer memory location has an address and holds a content.
  - The address is a numerical number (often expressed in hexadecimal)
- The value the variable stores is actually stored in the location assigned

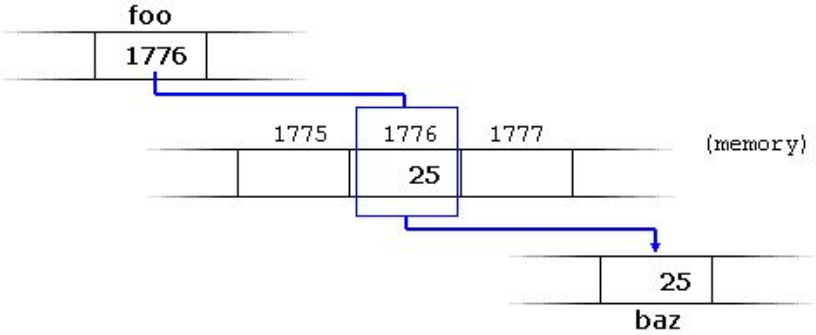
Pointers

- Pointer is a variable that stores a memory address.
- int* p (**or** int *p **or** int * p)
  - Initialize a pointer to an integer type
- &a
  - Address-of operator (of a variable a)
  - For example: **int* p = &a;**
- p
  - Address of where the pointer is pointing
- *p
  - Dereferencing operator i.e., refers to the value of the variable at the address stored in p

Pointers

- myvar = 25;
  foo = &myvar;
  bar = myvar;
- Dereferencing
  - baz = foo; // baz equal to foo (1776)
  - baz = *foo; // baz equal to value pointed to by foo (25)
- **& is the address-of operator, and can be read simply as "address of"**
- *** is the dereference operator, and can be read as "value pointed to by"**





Pointers to struct

- Use “->” to access the members of struct through pointers
  - Note: (*ptr).feet is equivalent to ptr->feet
  - Arrow operator makes it easy to keep track of which ones are variables, and which ones are pointers

- Mini-quiz from last week’s lecture:
  - What is the main difference between struct and class in C++?

```
struct Distance
{
        int feet ;
        int inch ;
};

int main()
{
        Distance d;
        // declare a pointer to Distance variable
        Distance* ptr;
        d.feet=8;
        d.inch=6;

        //store the address of d in p
        ptr = &d;
        cout<<"Distance="<< ptr->feet << "ft"<< ptr->inch << "inches";
        return 0;
}
```

Any questions so far?

Pass-by-Value

- Creates a local copy of variables

```
void add2 (int num)
{
    num = num + 2;
}

int main ()
{
    int a = 10;
    add2(a);
    cout << a;
}
```

Local variable of function “add2”

Local variable of function “main”

Pass-by-Value

- Creates a local copy of variables

```
void add2 (int num)
{
    num = num + 2;
}

int main ()
{
    int a = 10;
    add2(a);
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...

Pass-by-Value

- Creates a local copy of variables

```
void add2 (int num)
{
    num = num + 2;
}

int main ()
{
    int a = 10;
    add2(a);
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...
SOME ADDRESS	int num = 10

Pass-by-Value

- Creates a local copy of variables

```
void add2 (int num)
{
    num = num + 2;
}

int main ()
{
    int a = 10;
    add2(a);
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...
SOME ADDRESS	int num = 12

Pass-by-Value

- Creates a local copy of variables

```
void add2 (int num)
{
    num = num + 2;
}

int main ()
{
    int a = 10;
    add2(a);
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...
SOME ADDRESS	

Terminal

10

Pass-by-Pointers

- What happens in this case?

```
void add2 (int * num)
{
    *num = *num + 2;
}

int main ()
{
    int a = 10;
    add2( &a );
    cout << a;
}
```

Pass-by-Pointers

- What happens in this case?

```
void add2 (int * num)
{
    *num = *num + 2;
}

int main ()
{
    int a = 10;
    add2( &a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...

Pass-by-Pointers

- What happens in this case?

```
void add2 (int * num)
{
    *num = *num + 2;
}

int main ()
{
    int a = 10;
    add2( &a );
    cout << a;
}
```

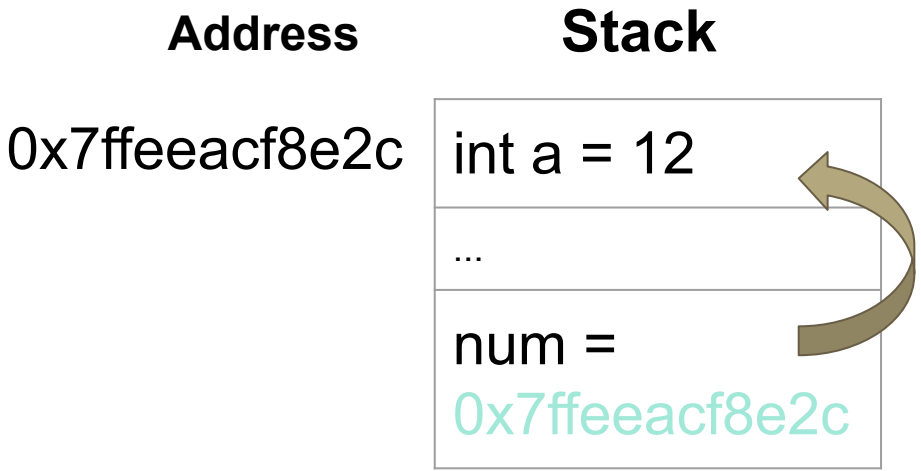
Address	Stack
0x7ffeeacf8e2c	int a = 10
	…
	num = 0x7ffeeacf8e2c

Pass-by-Pointers

- What happens in this case?

```
void add2 (int * num)
{
    *num = *num + 2;
}

int main ()
{
    int a = 10;
    add2( &a );
    cout << a;
}
```



Pass-by-Pointers

- What happens in this case?

```
void add2 (int * num)
{
    *num = *num + 2;
}

int main ()
{
    int a = 10;
    add2( &a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 12
	...

Terminal

12

Pass-by-Reference

- What happens in this case?

```
void add2 (int &num)
{
    num = num + 2 ;
}

int main ()
{
    int a = 10 ;
    add2( a );
    cout << a;
}
```

Pass-by-Reference

- What happens in this case?

```
void add2 (int &num)
{
    num = num + 2 ;
}

int main ()
{
    int a = 10 ;
    add2( a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...

Pass-by-Reference

- What happens in this case?

```
void add2 (int &num)
{
    num = num + 2 ;
}

int main ()
{
    int a = 10 ;
    add2( a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 10
	...

Pass-by-Reference

- What happens in this case?

```
void add2 (int &num)
{
    num = num + 2 ;
}

int main ()
{
    int a = 10 ;
    add2( a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 12
	...

Pass-by-Reference

- What happens in this case?

```
void add2 (int &num)
{
    num = num + 2 ;
}

int main ()
{
    int a = 10 ;
    add2( a );
    cout << a;
}
```

Address	Stack
0x7ffeeacf8e2c	int a = 12
	...

Terminal

12

Any questions?

Quiz

1. Given int a[] = {1, 2, 3}, what are the outputs of the following lines?
   - cout << a+2;
   - cout << *(a + 2);
   - cout << *a;
   - cout << *a + 2;
   - cout << *a[0];
2. How come we can pass an array name as an argument to a function and still be able to persist the change?

Exercise: Complete main.cpp and swap.cpp

int foo[5] = {1, 2, 3, 4, 5}

1. main.cpp
    a. Print addresses of each element in array foo
    b. Print values of each element in array foo using pointers
2. swap.cpp
    a. Swap elements in array foo using pointers
    b. Reverse array foo using pointers
3. swap.h

main.cpp (before the while loop)

```
#include<iostream>
#include "swap.h"

using namespace std;
int main(int argc, char const *argv[])
{

  int foo[5] = {1, 2, 3, 4, 5};

  cout << "Addresses of elements:" << endl;
  //TODO Print the addresses of array elements

  cout << endl;

  cout << "Elements:" << endl;;
  //TODO Print all the elements using pointers

  cout << endl;

  int a,b;
  int f;
  int flag = 1;
```

main.cpp (in the while loop)

When you type “0”, swap using pass-by-value

```
switch (f) {
  case 0:
    // Pass by Value
    // Compare the resultant array with the array you get after passing by pointer
    cout << "Before swapping" << endl;
    for(int i = 0;i<5;i++){
      cout<<foo[i]<<" ";
    }
    cout<<endl;

    swap(foo[a],foo[b]);

    cout << "\nAfter swapping" << endl;
    for(int i = 0;i<5;i++){
      cout<<foo[i]<<" ";
    }
    cout<<endl;
    break;
```

main.cpp (in the while loop)

When you type "1", swap using pass-by-pointer

```
case 1:
  // Pass by pointer
  cout << "Before swapping" << endl;
  for(int i = 0;i<5;i++){
    cout<<foo[i]<<" ";
  }
  cout<<endl;

  // TODO complete the function in swap.cpp file
  swap_by_pointers(&foo[a],&foo[b]);

  cout << "\nAfter swapping" << endl;
  for(int i = 0;i<5;i++){
    cout<<foo[i]<<" ";
  }
  cout<<endl;
  break;
```

main.cpp (after while loop)

Reverse the array foo using call-by-pointers

```
cout << "Reversing the array";
// Reverse your array
// TODO complete the function in swap.cpp file
reverse(foo,5);

cout << "\nAfter reversing" << endl;
for(int i = 0;i<5;i++){
  cout<<foo[i]<<" ";
}
cout<<endl;

return 0;
```

swap.cpp

```cpp
#include "swap.h"


// Function definitions
// Pass By Value
void swap(int n1, int n2) {
  int temp;
  temp = n1;
  n1 = n2;
  n2 = temp;
}

// Pass By Pointer
void swap_by_pointers(int *n1, int *n2) {
  // TODO Complete this function
}

// Function to reverse the array through pointers
void reverse(int array[], int array_size)
{
    // pointer1 pointing at the beginning of the array
    int *pointer1 = array;

    // pointer2 pointing at end of the array
    int *pointer2 = array + array_size - 1;
    // TODO Use the above swap function and update pointers to reverse your array
    //while (pointer1 < pointer2) {

    //}
```

What I expect to see when completing Recitation 2

```
Addresses of elements:
0x7ffeea51ce30 0x7ffeea51ce34 0x7ffeea51ce38 0x7ffeea51ce3c 0x7ffeea51ce40
Elements:
1 2 3 4 5
Enter indices of elements you want to swap?
First index
0
Second index
1
Enter 0 for pass-by-value, 1 for pass-by-pointer
1
Before swapping
1 2 3 4 5

After swapping
2 1 3 4 5

Press 1 to continue else press 0
0
Reversing the array
After reversing
5 4 3 1 2
```

Well,