CSCI 2270 – Data Structures

Recitation 2, Septmeber 2020

Objectives:

1. Pointers
2. Address-of Operator
3. Dereferencing Operator
4. Structs
5. Pointer to Structs
6. Pass-by-value vs Pass-by-pointer vs Pass-by-reference

1. Pointers

Every variable we declare in our code grabs a space in the memory. So it has an address in memory where it resides. A pointer is a variable that stores a memory address. By means of pointer variable, we can access the address and the value at that address.

2. Address-of Operator

Consider the following lines of code:

```
#include <iostream>
using namespace std;

int main()
{
	int a = 10;
	cout<< a << endl;
	cout<< &a << endl;
	return 0;
}
```

Output:

```
10
0x7ffccbbcd804
```

The first cout is quite straightforward. It is simply printing the value of a. But what is being printed by the second cout? It is the address of the variable a. The operator '&' is used to get the address of the variable a. Hence it is a referencing operator or address-of operator.

3. Dereferencing operator

Consider the following lines of code

```
#include <iostream>
using namespace std;

int main()
{
       int a = 10;

       // * here denote p is pointer variable
        int *p = &a;
        cout<< a <<endl;
        cout<< p <<endl;

       //* is used to dereference p
        cout<< *p << endl;
        return 0;
}
```

Output:

```
10
0x7ffccbbcd804
10
```

This code is a bit trickier than the previous one. Let's see what is happening in the code. Note that we are storing the address-of *a* in another variable p. Note the type of p. It is int* instead of just int. **Int *p** tells that p is a pointer variable (* denote it as a pointer) and it will store the address of an int variable.

The second cout prints p, the address of variable a. Now, what is the third cout doing? Remember p has the address of a. By putting a '*' before p, we are accessing the value stored at the position addressed by p. Hence, * is dereferencing the address p.

4. Structs

Why struct?
There are some instances wherein you need more than one variable to represent a complete object. As a student of CU, you should provide name, email, birthday, address to the college.

You could write code as follows:

```
std::string name;
std::string email;
```

```
int birthday;
std::string address;
```

But, the problem with this is you have 4 independent variables that are not grouped.

What is a struct?
C++ allows us to declare an aggregated(grouped) user-defined data type. This user-defined data type can, in turn, hold multiple variables of different data types (imagine an array that holds multiple values of different data types).

```
struct student
{
	std::string name;
	std::string email;
	int birthday;
	std::string address;
};
```

5. Pointer to a struct

We can have address (pointers) to any type of variable, even for structures.

```
#include <iostream>
using namespace std;

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;
}
```

Output
Why don't you try this one yourselves?
You may be wondering about '−>'. Recall to access members of a struct variable we use '.' operator (e.g. d.foot = 8). When we have a pointer to a struct variable to use the member variables we will use −> operator.

6. Pass-by-value vs Pass-by-pointer vs Pass-by-reference

Pass by Value

```
#include <iostream>
using namespace std;

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

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

What do you think the output will be? 12?
To your surprise, it will be just 10. When we pass a variable as an argument to a function in the system stack the function creates a local copy of the variable and performs the operation on that local copy. The caller function (main) has no knowledge of that local copy. Hence the change is local to the callee function (add2).

Pass by Pointers

```
#include <iostream>
using namespace std;

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

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

In this case, we are passing the address of a. The function will again create a local copy of the address. However, since both of these addresses are the same, they will refer to the variable a. Hence changing the value at the pointer will change the value of a and that change will be persisted.

Pass by Reference

```
#include <iostream>
using namespace std;

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

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

In C++, we can pass parameters to a function either by pointers or by reference. When you pass a parameter by reference, the parameter inside the function is an alias to the variable you passed from the outside. When you pass a variable by a pointer, you take the address of the variable and pass the address into the function.

Now examine the following code and try to find out why it is persisting the change?

```
void add2 (int a[], int len)
{
    for (int i=0; i<len; i++)
    a[i]+= 2;
}

int main ()
{
    int a[] = { 1 , 2 , 3 };
    add2( a, 3 );
    for ( int i=0;i< 3;i++)
      cout << a[i] << endl ;
}
```

Is the previous one similar/different to the one given below?

```
void add2 ( int *a, int len)
{
    for ( int i=0;i<len;i++)
    a[i]+= 2 ;
}

int main ()
{
    int a[] = { 1 , 2 , 3 };
    add2( &a[ 0 ], 3 );
    for ( int i=0; i<3; i++)
      cout << a[i] << endl ;
}
```

Tip!
Pass By Value
If I send a fax to someone to sign, he creates a local copy i.e prints it and then signs. The person makes changes to his local copy that he printed out(The signature won't be reflected in my original copy). He has to scan it and send it back, for me to see his signed document.

Pass by reference
Now consider I send my address to the person who is required to sign. The person comes to my place and then signs. In this case, the person is modifying my original document.

Quiz

1. Consider an array **int a[] = {1, 2, 3}.** What is the output for the following?
    a. cout << a+2;
    b. cout << *(a+2);
    c. cout << *a;
    d. 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

Your zipped folder for this Lab will have a main.cpp, swap.cpp and swap.h files. Follow the TODOs in these files to complete your Recitation exercise!