CSCI 2270 – Data Structures

Recitation 2, Fall 2022

Objectives

1. Pointers
2. Address-of Operator (&)
3. Dereferencing Operator
4. Structs
5. Pointer to structs.

1. Pointers

To understand pointers, we investigate how a C++ program operates on a Computer's memory. For a C++ program, a computer memory is like a succession of memory cells, each one byte in size, and each with a unique address (hexa-decimal). These single byte memory cells are ordered such that datatypes with larger memory footprint (integer, float, long etc..) occupy memory cells that have consecutive addresses.

This way, an integer (4 bytes) can be stored in contiguous memory addresses starting from 1330 to 1333. And the unique memory address for this integer is identified as 1330.

For an array of integers of size 10 stored in contiguous memory addresses starting from 1401 to 1440, the array location is easily identified by its unique memory address 1401. The 0^{th} indexed element of the array is stored in the memory block 1401-1404, the 1^{st} indexed element in 1405-1408 and so on. It is here we need Pointers.

A pointer is a variable which stores a memory address of a variable. In the above example of an array, a pointer can be declared to store the address of the array (1401). Using this address, the C++ program can access the contents of the array by calculating the relative position from the starting address (1401).

2. Address-of Operator (&)

For the curious, you can get the physical memory address location of any variable by prefixing the variable with an ampersand (&) operator. This is called an *Address-of or Referencing operator*. The address as shown in the output is in a hexadecimal format.

Filename: address_of.cpp

```cpp
#include<iostream>

using namespace std;

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

Output

```
10
0x7ffffddca5c4
```

3. Dereferencing operator (*)

As we have mentioned in the section “Pointers”, they are just variables which are used to store the address of other variables. Colloquially, the Pointers are said to “point to” the variable whose address they store.

It gets its name as Pointers can be used to directly access the variable whose address it points to. This is done by prefixing the pointer variable with the ***dereferencing operator*** (*).

Pointer variables are declared differently compared to regular variables. We know that we can fetch the address of a variable by prefixing it with an ampersand (&), and we have now learnt that the pointers store the addresses of a variable. The code below shows how the address of **a** is stored in a pointer variable **p,** and **q**.

For the keen eye, one can notice that both the pointers **p** and **q** “point to” the address of the same variable **a**. Thus, it is possible to have multiple pointers “point to” the SAME variable.

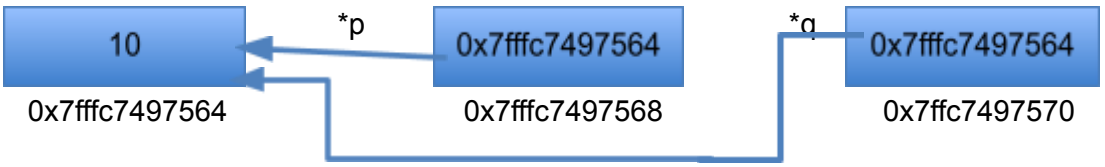
```cpp
#include <iostream>
using namespace std;

int main()
{
    int a = 10;
    // * here denote p is pointer variable
    int *p;
    int *q;
    p = &a;
    q = &a;
    cout<< a <<endl;
    cout<< p <<endl;
    cout << q << endl;
    //* is used to dereference p
    cout << "Address of the pointer p: " << &p << endl;
    cout << "Address stored in pointer p: " << p << endl;
    cout << "The value of the pointer p is pointing to: " << *p << endl;
    cout << "Address of the pointer q: " << &q << endl;
    return 0;
}
```

Output

```
10
0x7fffc7497564
0x7fffc7497564
Address of the pointer p: 0x7fffc7497568
Address stored in pointer p: 0x7fffc7497564
The value of the pointer p is pointing to: 10
Address of the pointer q: 0x7fffc7497570
```

For better understanding, visual representation of the above program is shown below. The data within the box is the data stored in the specific memory location. The variables **a**, **p**, and **q** on accessing provides the data within the memory, and the characters below the boxes represent the physical memory address location.

a p q



4. Structs

Sometimes, to represent a real-world object, simple datatypes are not enough. To represent a Student, we need some (if not all) of the following: age, gender, date-of-birth, name, address etc.

You could write code as follows:

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

This becomes increasingly complex if we would like to store multiple Student representations. One way to solve this is by using aggregated(grouped) user-defined datatype called **Struct**. This datatype can hold different datatypes grouped under a common variable of type Struct.

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

5. Pointer to struct

Pointers apply not only to simple datatypes, but aggregated datatypes as well.
We can have addresses (pointers) to any type of variable, even for structures. The following code shows how pointers can point to an aggregated datatype, as well as ways to assign values to struct and accessing them directly, as well as through a pointer.

Notice the **cout** statements. To access members of the struct variable we use **dot (.)** operator, and to access the members via a pointer variable, we use **-> (dash followed by a greater than sign)**.

```cpp
#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 << "Variable access" << endl;
    cout << "Distance= "<< d.feet << "ft "<< d.inch << "inches" << endl;
    cout << "Pointer access" << endl;
    cout << "Distance= " << ptr->feet << "ft " << ptr->inch << "inches" << endl;
    return 0;
}
```

Output

```
Variable access: Distance= 8ft 6inches
Pointer access: Distance= 8ft 6inches
```

Exercise

Create a struct named TaxInfo with the salary and tax percentage values of the salary. Write a C++ program to take inputs of salary (in dollars) and tax rate (in percent) from the user as arguments and assign those values to a TaxInfo variable. Also, print the individual values, tax incurred (Tax Calculation) of that person in the standard output.

To calculate the tax you need to multiply the salary with the tax ratio (divide the tax rate by 100).

Constraint: You can only use pointer access for calculation and printing the values.
Complete the **TODOs** mentioned in the folder.

Example Output:

```
hunarjain09@Hunars-MacBook-Pro soln % ./a.out 150 10
The salary is : 150
The tax rate is : 10
The tax incurred is : 15
hunarjain09@Hunars-MacBook-Pro soln %
```

References

- https://www.cplusplus.com/doc/tutorial/pointers/