LAB 15: Pointers

Objective(s):

To Understand about:

1. Able to use Pointers
2. Able to use pointers in functions

CLOs: CLO1, CLO4

Introduction

Pointers are powerful features of C++ that differentiates it from other programming languages like Java and Python.

Pointers are used in C++ program to access the memory and manipulate the address.

Address in C++

To understand pointers, you should first know how data is stored on the computer. Each variable you create in your program is assigned a location in the computer's memory. The value the variable stores is actually stored in the location assigned. To know where the data is stored, C++ has an & operator. The & (reference) operator gives you the address occupied by a variable. If var is a variable then, &var gives the address of that variable.

Example : Address in C++

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

int main()
{
   int var1 = 3;
   int var2 = 24;
   int var3 = 17;
   cout << &var1 << endl;
   cout << &var2 << endl;
   cout << &var3 << endl;
}
```

Output

```
0x7fff5fbff8ac
0x7fff5fbff8a8
0x7fff5fbff8a4
```

Note: You may not get the same result on your system.

The 0x in the beginning represents the address is in hexadecimal form.

Notice that first address differs from second by 4-bytes and second address differs from third by 4-bytes.

This is because the size of integer (variable of type int) is 4 bytes in 64-bit system.

Pointers Variables

C++ gives you the power to manipulate the data in the computer's memory directly. You can assign and de-assign any space in the memory as you wish. This is done using Pointer variables.

Pointers variables are variables that points to a specific address in the memory pointed by another variable.

How to declare a pointer?

```
int *p;
    OR,
int* p;
```

The statement above defines a pointer variable p. It holds the memory address

The asterisk is a dereference operator which means pointer to.

Here, pointer p is a pointer to int, i.e., it is pointing to an integer value in the memory address.

Reference operator (&) and Deference operator (*)

Reference operator (&) as discussed above gives the address of a variable.

To get the value stored in the memory address, we use the dereference operator (*).

For example: If a number variable is stored in the memory address 0x123, and it contains a value 5.

The reference (&) operator gives the value 0x123, while the dereference (*) operator gives the value 5.

Note: The (*) sign used in the declaration of C++ pointer is not the dereference pointer. It is just a similar notation that creates a pointer.

Example: C++ Pointers

C++ Program to demonstrate the working of pointer.

```
#include <iostream>
using namespace std;
int main() {
   int *pc, c;

   c = 5;
   cout << "Address of c (&c): " << &c << endl;
   cout << "Value of c (c): " << c << endl << endl;

   pc = &c;    // Pointer pc holds the memory address of variable c
   cout << "Address that pointer pc holds (pc): "<< pc << endl;
   cout << "Content of the address pointer pc holds (*pc): " << *pc << endl << endl;

   c = 11;    // The content inside memory address &c is changed from 5 to 11.
   cout << "Address pointer pc holds (pc): " << pc << endl;
   cout << "Content of the address pointer pc holds (*pc): " << *pc << endl << endl;
```

```
    *pc = 2;
    cout << "Address of c (&c): " << &c << endl;
    cout << "Value of c (c): " << c << endl << endl;

    return 0;
}
```

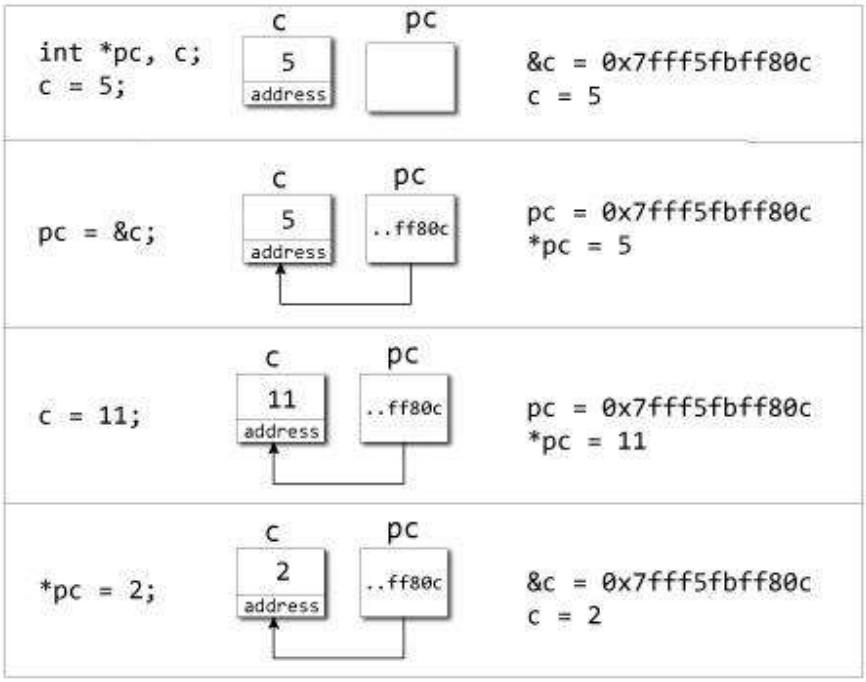
Output

```
Address of c (&c): 0x7fff5fbff80c
Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff5fbff80c
Content of the address pointer pc holds (*pc): 5

Address pointer pc holds (pc): 0x7fff5fbff80c
Content of the address pointer pc holds (*pc): 11

Address of c (&c): 0x7fff5fbff80c
Value of c (c): 2
```



Explanation of program

- When c = 5; the value 5 is stored in the address of variable c - 0x7fff5fbff8c.
- When pc = &c; the pointer pc holds the address of c - 0x7fff5fbff8c, and the expression (dereference operator) *pc outputs the value stored in that address, 5.
- When c = 11; since the address pointer pc holds is the same as c - 0x7fff5fbff8c, change in the value of c is also reflected when the expression *pc is executed, which now outputs 11.

- When *pc = 2; it changes the content of the address stored by pc - 0x7fff5fbff8c. This is changed from 11 to 2. So, when we print the value of c, the value is 2 as well.

Common mistakes when working with pointers

Suppose, you want pointer pc to point to the address of c. Then,

```
int c, *pc;
pc=c;  /* Wrong! pc is address whereas, c is not an address. */
*pc=&c; /* Wrong! *pc is the value pointed by address whereas, %amp;c is an address. */
pc=&c; /* Correct! pc is an address and, %amp;pc is also an address. */
*pc=c; /* Correct! *pc is the value pointed by address and, c is also a value. */
```

Passing pointer values in C++

Like any other type, a pointer may be passed as an argument to a function:

```
void fn(int* pnArg)
{
  *pnArg = 10;
}
void parent(void)
{
   int n = 0;
   fn(&n);      // this passes the address of i
                // now the value of n is 10
}
```

In this case, the address of n is passed to the function fn() rather than the value of n. The significance of this difference is apparent when you consider the assignment within fn().

Suppose n is located at address 0x100. Rather than the value 10, the call fn(&n) passes the value 0x100. Within fn(), the assignment *pnArg = 10 stores the value 10 in the int variable located at location 0x100, thereby overwriting the value 0. Upon returning to parent(), the value of n is 10 because n is just another name for 0x100.

Instead of a regular value or even a reference, a function can return a pointer. You can start to specify this by typing the * operator on the left side of the function's name. Here is an example:

```
double * getSalary()

{

}
```

Then, use the body of the function to define it. Before the closing curly bracket of the function, remember to return a pointer to the return value. Here is an example:

```
double * getSalary()

{
   double salary = 26.48;
   double *hourlySalary = &salary;
   return hourlySalary;
}
```

Because a pointer by defining is a reference to the address where a variable resides, when a function is defined as returning a pointer, you can also return a reference to the appropriate type. Here is an example:

```
double * getSalary()

{
   double salary = 26.48;
   return &salary;
}
```

Dynamic Memory Allocation:
Dynamic memory allocation in C/C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack.

One use of dynamically allocated memory is to allocate memory of variable size which is not possible with compiler allocated memory except variable length arrays.
The most important use is flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need and whenever we don't need anymore. There are many cases where this flexibility helps.

C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory. C++ supports these functions and also has two operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

new operator
The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Syntax to use new operator:

To allocate memory of any data type, the syntax is:

pointer-variable = new data-type;

Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class.

Example:
// Pointer initialized with NULL

// Then request memory for the variable

int *p = NULL;

p = new int;

OR

// Combine declaration of pointer

// and their assignment

int *p = new int;

Allocate block of memory: new operator is also used to allocate a block(an array) of memory of type data-type.

pointer-variable = new data-type[size];

where size(a variable) specifies the number of elements in an array.

Example:

int *p = new int[10]

Dynamically allocates memory for 10 integers continuously of type int and returns pointer to the first element of the sequence, which is assigned to p(a pointer). p[0] refers to first element, p[1] refers to second element and so on.

delete operator

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

Syntax:

// Release memory pointed by pointer-variable

delete pointer-variable;

Here, pointer-variable is the pointer that points to the data object created by new.

Examples:

```
delete p;

delete q;
```

To free the dynamically allocated array pointed by pointer-variable, use following form of delete:

// Release block of memory

// pointed by pointer-variable

delete[] pointer-variable;

Lab Tasks:

Task 1
Write a program that swaps two values by passing pointers as arguments to function.

Task 2
Write a program that take five input by user in an array and pass it to function. Find the smallest number in an array using pointers.

Task 3
Write a program to find the sum of array and find factorial using sum of array using pointers.