# Department of Electrical Engineering

**CS212**

**Object Oriented Programming**



# Lab 3: Pointers

**Class**: BEE - 12C

**Date**: October 4th, 2021

**Time**: Monday (1400 – 1700)

**Name**: Muhammad Umer

**CMS ID**: 345834

## Introduction

A procedural program is written as a list of instructions, telling the computer, step-by-step, what to do: Open a file, read a number, multiply by 4, display something. Program units include the main or program block, subroutines, functions, procedures; file scoping; includes/modules; libraries.

Procedural programming is fine for small projects. It is the most natural way to tell a computer what to do, and the computer processor's own language, machine code, is procedural, so the translation of the procedural high-level language into machine code is straightforward and efficient. What is more, procedural programming has a built-in way of splitting big lists of instructions into smaller lists: the function.

## Objectives

The objective of this lab is review of:

* Pointers

## Tools/Software Requirement

* Microsoft Visual Studio

## Description

You are required to complete the given tasks using procedural language C. The idea is to review the procedural language C that you have already understood. Remember in structured approach functionality is identified and functions are created.

# What is a pointer?

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type. The unary or monadic operator **&** gives the address of a variable. The indirection or dereference operator **\*** gives the ``contents of an object pointed to by a pointer''. To declare a pointer to a variable does:

int \*pointer;

Consider the effect of the following code:

int x = 1, y = 2;

int \*ip;

ip = &x;

y = \*ip;

x = ip;

\*ip = 3;

It is worth considering what is going on at the **machine level** in memory to fully understand how pointer work.

Assume for the sake of this discussion that variable x resides at memory location 100, y at 200 and ip at 1000. **Note** A pointer is a variable and thus its values need to be stored somewhere. It is the nature of the pointers value that is new.

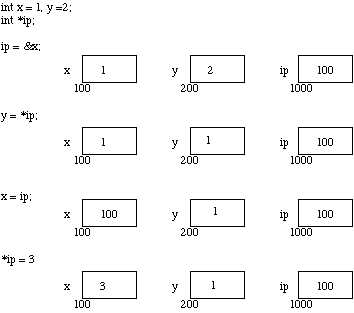


Figure 1

**Fig.1 Pointer, Variables and Memory** Now the assignments x = 1 and y = 2 obviously load these values into the variables. ip is declared to be a pointer to an integer and is assigned to the address of x (&x). So ip gets loaded with the value 100.

Next y gets assigned to the contents of ip. In this example ip currently points to memory location 100 -- the location of x. So y gets assigned to the values of x -- which is 1.

We have already seen that C is not too fussy about assigning values of different type. Thus it is perfectly **legal** (although not all that common) to assign the current value of ip to x. The value of ip at this instant is 100.

Finally we can assign a value to the contents of a pointer (\*ip)

int \*ip;

\*ip = 100;

will generate an error (program crash!). The correct use is:

int \*ip;

int x;

ip = &x;

\*ip = 100;

We can do integer arithmetic on a pointer:

float \*flp, \*flq;

\*flp = \*flp + 10;

++\*flp;

(\*flp)++;

flq = flp;

***A pointer to any variable type is an address in memory -- which is an integer address. A pointer is definitely NOT an integer.***

The reason we associate a pointer to a data type is so that it knows how many bytes the data is stored in. When we increment a pointer we increase the pointer by one ``block'' memory.

So for a character pointer ++ch\_ptr adds 1 byte to the address.

For an integer or float ++ip or ++flp adds 4 bytes to the address.

Consider a float variable (fl) and a pointer to a float (flp) as shown in Fig.2.

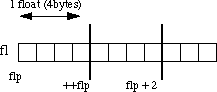


Figure 2

**Pointer Arithmetic** Assume that flp points to fl then if we increment the pointer (++flp) it moves to the position shown 4 bytes on. If on the other hand we added 2 to the pointer then it moves 2 **float positions** i.e 8 bytes as shown in the Figure.

**Assigning a Variable's Memory Address to a Pointer**

Before we can assign a memory address to a pointer, we need to declare one. Declaring a pointer in C++ is as simple as to declare any other variable with one single difference. Asterix symbol " \* " needs to be add and located after variable type and before a variable name. One rule has to be followed when assigning memory address to a pointer: pointer type has to match with variable type it will point to. One exception is a pointer to void, which can handle different types of variables it will point to. To declare a pointer pMark of type unsigned short int a following syntax is to be used:

**Lab Practice 2**

#include <iostream>

using namespace std;

int main()

{

    unsigned short int \*pPointer = 0;

    unsigned short int twoInt = 35698;

    unsigned short int oneInt = 77;

    pPointer = &twoInt;

    cout << "\t\tpPointer's memory address: " << &pPointer << endl;

    cout << "Integer one's Int memory address: " << &oneInt << " Integer value: " << oneInt << endl;

    cout << "Integer two's Int memory address: " << &twoInt << " Integer value: " << twoInt << endl;

    cout << "pPointer is pointing to memory address: " << pPointer << " Integer value: " << \*pPointer << endl;

    return 0;

}

**Terminal Output**

                pPointer's memory address: 0x2c739ffd78

Integer one's Int memory address: 0x2c739ffd74 Integer value: 77

Integer two's Int memory address: 0x2c739ffd76 Integer value: 35698

pPointer is pointing to memory address: 0x2c739ffd76 Integer value: 35698

**Accessing the Value at the Memory Address held by a Pointer**

As you could see in the previous example a pointer pMark truly holds a value memory address of an oneInt. Process accessing a variable's value by a pointer is called indirection, since the value of variable is accessed indirectly. Value of oneInt can be now indirectly accessed with a use of pPointer pointer. To do that we need to dereference a pointer with dereference operator “ \* “ which needs to be placed before a pointer variable name

**Lab Practice 3**

#include <iostream>

using namespace std;

int main() {

    unsigned short int myInt = 99;

    unsigned short int \*pMark = 0;

    cout << myInt << endl;

    pMark = &myInt;

    cout << \*pMark << endl;

    return 0;

}

**Terminal Output**

99

99

**Manipulating Data with Pointers**

Same as accessing the value at the memory address held by pointer by indirection, the indirection can also be used to manipulate variable's value. Assigning a value to a dereferenced pointer will indirectly change a value of a variable the pointer is pointing to. The following example illustrates simple manipulation of data with pointers:

**Lab Practice 4**

#include <iostream>

using namespace std;

int main() {

    unsigned short int myInt = 99;

    unsigned short int \* pMark = 0;

    cout << myInt << endl;

    pMark = &myInt;

    \*pMark = 11;

    cout << "\*pMark:\t" << \*pMark << "\nmyInt:\t" << myInt << endl;

    return 0;

}

**Terminal Output**

99

\*pMark: 11

myInt:  11

**Lab Tasks**

* **Task 1**

Consider the following program and answer the questions:

#include <iostream>

using namespace std;

int main() {

    int a, \*pa;  // Statement 1

    pa = &a;     // Statement 2

    cout << "pa = &a --> pa = " << pa << endl;

    pa = pa + 1;  // Statement 3

    cout << "pa = pa + 1 --> pa = " << pa << endl;

    pa = pa + 3;  // Statement 4

    cout << "pa = pa + 3 --> pa = " << pa << endl;

    pa = pa - 1;  // Statement 5

    cout << "pa = pa - 1 --> pa = " << pa << endl;

  return 0;

}

**Terminal Output**

pa = &a --> pa = 0x42c67ff844

pa = pa + 1 --> pa = 0x42c67ff848

pa = pa + 3 --> pa = 0x42c67ff854

pa = pa - 1 --> pa = 0x42c67ff850

1. **Why the memory address stored in pointer “pa” vary by 4?**

Considering the case of an integer address, it takes up a space of 4 bits in a memory register and hence causes each variable’s address in the program to increment by 4 bits, which is essentially the value of the ***sizeof*** of the pointed element.

1. **Will the address still vary by 4 if one changes the data type of the above mentioned code from “int” to “long”? Explain your answer.**

No, in C++, long takes up the same memory space, i.e., 4 bytes, as an ordinary integer so changing from int to long would not have any effect. Unless we use “long long int” or some other data type, the address will be varying by a value of 4.

1. **If we try to multiply the address pointed to by “pa” then what will happen? Is this logically or programmatically correct?**

C++ does not support the multiplication of address hence this is programmatically incorrect as multiplication is not defined for pointers and will cause the original code to crash.

* **Task 2**

Write a code to find the memory in bytes occupied by int, long, double, float and char.

#include <iostream>

using namespace std;

int main()

{

    cout << "Size of Char: " << sizeof(char) << " byte" << endl;

    cout << "Size of Int: " << sizeof(int) << " bytes" << endl;

    cout << "Size of Long: " << sizeof(long int) << " bytes" << endl;

    cout << "Size of Float: " << sizeof(float) << " bytes" << endl;

    cout << "Size of Double: " << sizeof(double) << " bytes" << endl;

    return 0;

}

**Terminal Output**

Size of Char: 1 byte

Size of Int: 4 bytes

Size of Long: 4 bytes

Size of Float: 4 bytes

Size of Double: 8 bytes

* **Task 3**

Write down a program which takes two integers and swap their values? You may create a function swap which takes int a and int b as arguments then swap their values in such a way that it assigns initial value of a to b and initial value of b to a.

#include <iostream>

using namespace std;

int main()

{

    int a, b, z;

    cout << "Enter two variables to swap:\n" << "a: ";

    cin >> a;

    cout << "b: ";

    cin >> b;

    cout << "Before swapping:\n a: " << a << "\tb: " << b << endl;

    // Procedural Approach

    z = b;

    b = a;

    a = z;

    cout << "After swapping:\n a: " << a << "\tb: " << b << endl;

    return 0;

}

**Terminal Output**

Enter two variables to swap:

a: 7

b: 213

Before swapping:

 a: 7   b: 213

After swapping:

 a: 213 b: 7

* **Task 4**

Write down a program which takes two integers and swap their values using pass by reference.

#include <iostream>

using namespace std;

void passByReference(int \*x, int \*y) {

    int z = \*y;

    \*y = \*x;

    \*x = z;

}

int main()

{

    int a, b, z;

    cout << "Enter two variables to swap:\n" << "a: ";

    cin >> a;

    cout << "b: ";

    cin >> b;

    cout << "\tPass By Reference Swapping\n";

    cout << "Before swapping:\n a: " << a << "\tb: " << b << endl;

    passByReference(&a, &b);

    cout << "After swapping:\n a: " << a << "\tb: " << b << endl;

    return 0;

}

**Terminal Output**

Enter two variables to swap:

a: 78

b: 123

Pass By Reference Swapping

Before swapping:

 a: 78  b: 123

After swapping:

 a: 123 b: 78