|  |  |
| --- | --- |
| **Name** |  |
| **CMS ID** |  |
| **Marks** |  |
| **Date** |  |

**Objective:** To become familiar with Pointers

* 1. **Pointers**

Variables are locations in the computer's memory which can be accessed by their identifier (their name). This way, the program does not need to care about the physical address of the data in memory; it simply uses the identifier whenever it needs to refer to the variable.  
  
For a C++ program, the memory of a computer is like a succession of memory cells, each one byte in size, and each with a unique address. These single-byte memory cells are ordered in a way that allows data representations larger than one byte to occupy memory cells that have consecutive addresses.  
  
This way, each cell can be easily located in the memory by means of its unique address. For example, the memory cell with the address 1776 always follows immediately after the cell with address 1775 and precedes the one with 1777, and is exactly one thousand cells after 776 and exactly one thousand cells before 2776.  
  
When a variable is declared, the memory needed to store its value is assigned a specific location in memory (its memory address). Generally, C++ programs do not actively decide the exact memory addresses where its variables are stored. Fortunately, that task is left to the environment where the program is run - generally, an operating system that decides the particular memory locations on runtime. However, it may be useful for a program to be able to obtain the address of a variable during runtime in order to access data cells that are at a certain position relative to it.

The data types that you have worked with until now were mostly int, bool, char, and double. Even though these data types are sufficient to solve just about any problem, situations occur when these data types are not adequate to solve a particular problem.

* 1. **Address operator (&)**

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as address-of operator. For example:

cout<<&num;

you can store the address of a variable in a specific variable. Such as

ptr = &num;

The variable that stores the address of another variable (like ptr in the previous example) is what in C++ is called a pointer. Pointers are a very powerful feature of the language that has many uses in lower level programming. A bit later, we will see how to declare and use pointers.

**1.3 Dereference operator(\*)**

As just seen, a variable which stores the address of another variable is called a pointer. Pointers are said to "point to" the variable whose address they store.  
  
An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the dereference operator (\*). The operator itself can be read as "value pointed to by".

int \*ptr\_int, int\_num1=8, int\_num2;

ptr\_int = &int\_num1; // ptr\_int stores the address of int\_num1  
int\_num2 = \*ptr\_int; //copies the value of int\_num1 into int\_num2

Lets try an example of accessing variables through pointers

**Example No: 01**

#include <iostream>

using namespace std;

int main ()

{

int firstvalue, secondvalue;

int \* mypointer;

mypointer = &firstvalue;

\*mypointer = 10;

mypointer = &secondvalue;

\*mypointer = 20;

cout << "firstvalue is " << firstvalue << '\n';

cout << "secondvalue is " << secondvalue << '\n';

return 0;

}

|  |
| --- |
| **Output:** |

**1.4 Pointers and arrays**

The concept of arrays is related to that of pointers. In fact, arrays work very much like pointers to their first elements, and, actually, an array can always be implicitly converted to the pointer of the proper type. For example, consider these two declarations:

int myarray [20];

int \* mypointer;

The following assignment operation would be valid:

mypointer = myarray;// or mypointer = &myarray[0];

After that, mypointer and myarray would be equivalent and would have very similar properties. The main difference being that mypointer can be assigned a different address, whereas myarray can never be assigned anything, and will always represent the same block of 20 elements of type int. Therefore, the following assignment would not be valid:

|  |
| --- |
|  |

myarray = mypointer;

Let's see an example that mixes arrays and pointers:

**Example No:02**

#include <iostream>

using namespace std;

int main ()

{

int numbers[5];

int \* p;

p = numbers; \*p = 10;

p++; \*p = 20;

p = &numbers[2]; \*p = 30;

p = numbers + 3; \*p = 40;

p = numbers; \*(p+4) = 50;

for (int n=0; n<5; n++)

cout << numbers[n] << ", ";

return 0;

}

|  |
| --- |
| **Output:** |

Pointers and arrays support the same set of operations, with the same meaning for both. The main difference being that pointers can be assigned new addresses, while arrays cannot.  
  
In arrays, brackets ([]) were explained as specifying the index of an element of the array. Well, in fact these brackets are a dereferencing operator known as *offset operator*. They dereference the variable they follow just as \* does, but they also add the number between brackets to the address being dereferenced. For example:

a[5] = 0; // a [offset of 5] = 0

\*(a+5) = 0; // pointed to by (a+5) = 0

These two expressions are equivalent and valid, not only if a is a pointer, but also if a is an array. Remember that if an array, its name can be used just like a pointer to its first element.

**1.5 Pointers and functions**

C++ allows operations with pointers to functions. The typical use of this is for passing a function as an argument to another function. Pointers to functions are declared with the same syntax as a regular function declaration, except that the name of the function is enclosed between parentheses () and an asterisk (\*) is inserted before the name:

**Example No:03**

#include <iostream>

using namespace std;

int addition (int \*a, int \*b)

{ \*a = 18; \*b = 10; return (\*a+\*b); }

int subtraction (int a, int b)

{ a = 18; b = 10; return (a-b); }

int main ()

{

int m = 6,n = 8;

cout<<m<<’-’<<n<<” = “<<subtraction(m,n)<<endl;

cout<<m<<” & ”<<n<<” after function call are ”<<m<<”&”<<n<<endl;

cout<<m<<’+’<<n<<” = “<<addition(&m,&n)<<endl;

cout<<m<<” & ”<<n<<” after function call are ”<<m<<”&”<<n<<endl;

return 0;

}

|  |
| --- |
| **Output:** |

Programming Exercise:

a. Write a program to find the factorial of a number using pointers.

b. Write a program that defines two arrays named array1[2][2] and array2[2][2], stores some values in them, and assign base address of array1 and array2 to ptr1 and ptr2 respectively. It should then apply matrix multiplication of both arrays using pointers and stores them in another array named array3[2][2] whose base address is stored in ptr3.

c. Write a program that defines two arrays named array\_1D[12] and array\_2D[3][4]. Stores the base addresses of array\_1D and array\_2D in ptr1 and ptr2 respectively. It should then ask user to enter the values of elements of array\_1D whereas leaving array\_2D empty. It should then define a function that gets base addresses of both the arrays using ptr1 and ptr2 and copy the elements of array\_1D into array\_2D using pointers only.

d. **Work on your semester project.**