Advanced Programming

Pointers in C++

Dr. Zeyad Safaa Younus Saffawi

Pointers in C++

- **Pointers** are an important concept in **C++ programming**.
- **Pointers** are **used** when we need to **work directly** with **memory addresses**.
- Some **tasks** in C++ become **easier using pointers**, such as:
  - Working with **arrays**
  - **Dynamic memory allocation**
  - **Passing variables** efficiently to **functions**
- Every **variable** in a program is **stored** in **memory**, and each **memory location** has a **unique address**.
- We can **access** the **address** of a **variable** using the **ampersand operator (&)**.

Example: Getting the Address of a Variable

```
#include <iostream>
using namespace std;
int main()
{
   int var1;
   char var2[10];
   cout << "Address of var1 variable: ";
   cout << &var1 << endl;
   cout << "Address of var2 variable: ";
   cout << &var2 << endl;
   return 0;
}
```

Output:

```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

Note: The program prints the **memory address** of each variable, **not** its **value**.

What is a Pointer?

- A **Pointer** is **a variable** that **stores** the **address of another variable**.
- Instead of storing a value directly, the pointer stores the **location of that value in memory**.
- **Pointer Declaration**

  The general form of declaring a pointer is:

  type * pointer_name;

- Where:
  - **type**: data type of the variable being pointed to
  - * : indicates that the variable is a pointer

Examples of Pointer Declarations

```
int *ip;          // pointer to an integer
double *dp;       // pointer to a double
float *fp;        // pointer to a float
char *ch;         // pointer to a character
```

- All pointers store **memory addresses**, but they differ in the **type of data they point to**.

Using Pointers

- Using pointers usually involves three steps:
  1. Declare a pointer variable
  2. Assign the address of a variable to the pointer
  3. Access the value stored at that address

Example:

```
#include <iostream>
using namespace std;
int main()
{
    int var = 20;
    int *ip;
    ip = &var;
    cout << "Value of var: " << var << endl;
    cout << "Address stored in ip: " << ip << endl;
    cout << "Value using pointer (*ip): " << *ip << endl;
    return 0;
}
```

Expression	Meaning
&var	address of variable
ip	pointer storing the address
*ip	value stored at that address

Output:

```
Value of var: 20
Address stored in ip: 0xbfc601ac
Value using pointer (*ip): 20
```

Null Pointer

- A **Null Pointer** is a **pointer** that does not **point to any valid memory location**.
- It is good practice to **initialize pointers** with **NULL** if they are **not assigned to any address**.
- Example:

```
#include <iostream>
using namespace std;
int main()
{
    int *ptr = NULL;
    cout << "The value of ptr is " << ptr;
    return 0;
}
```

Output:

- The value of ptr is 0

Pointer Arithmetic

- Since a pointer contains a memory address, **arithmetic operations** can be performed on it.
- Common operations:
    - ++
    - --
    - +
    - -

Example:

- If an **integer pointer** points to **address 1000** and the **size of an integer** is **4 bytes**:

ptr++

- The **pointer** will move to address **1004**, which is the **next integer location**.

Incrementing a Pointer

- Pointers can be used to access array elements.

```
#include <iostream>
using namespace std;
int main()
{
   int var[3] = {10,100,200};
   int *ptr;
   ptr = var;
   for(int i=0;i<3;i++)
   {
      cout<<"Address of var["<<i<<"] = "<<ptr<<endl;
      cout<<"Value of var["<<i<<"] = "<<*ptr<<endl;
      ptr++;
   }
   return 0;
}
```

- The pointer moves to the **next array element** each time it is incremented.

Decrementing a Pointer

- A pointer can also move **backward** in memory.

```
#include <iostream>
using namespace std;
int main()
{
    int var[3] = {10,100,200};
    int *ptr;
    ptr = &var[3-1];
    for(int i=3;i>0;i--)
    {
        cout<<"Address = "<<ptr<<endl;
        cout<<"Value = "<<*ptr<<endl;
        ptr--;
    }
    return 0;
}
```


The program starts from the **last element of the array** and moves backward.

Pointer Comparison

Pointers can be compared using **relational operators** such as:

- **==**
- **<**
- **>**
- **<=**
- **>=**

This comparison is meaningful when pointers refer to **elements of the same array**.

Example of relational operators

```
#include <iostream>
using namespace std;
int main()
{
   int arr[3] = {10, 20, 30};
   int *p1 = &arr[0];   // pointer to first element
   int *p2 = &arr[2];   // pointer to last element
   if(p1 == p2)
      cout << "p1 is equal to p2" << endl;
   if(p1 < p2)
      cout << "p1 is less than p2" << endl;
   if(p2 > p1)
      cout << "p2 is greater than p1" << endl;
if(p1 <= p2)
      cout << "p1 is less than or equal to p2" << endl;
   if(p2 >= p1)
      cout << "p2 is greater than or equal to p1" << endl;
   return 0;
}
```

Output:

- p1 is less than p2
- p2 is greater than p1
- p1 is less than or equal to p2
- p2 is greater than or equal to p1

Important operators:

Operator	Meaning
&	get address
*	access value from address
NULL	pointer that does not point anywhere
++	move to next memory location
--	move to previous memory location

Pointers are widely **used** in:

- **arrays**
- **dynamic memory allocation**
- **efficient function calls**

References

- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.