Problem Solving and Programming 2

Lecture Three

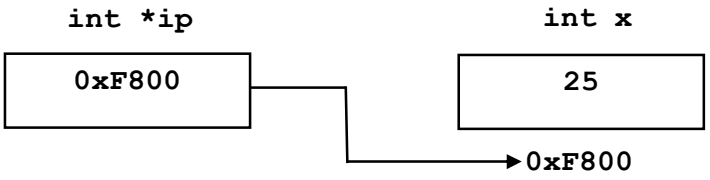
Dealing with Pointers in C++

- A **pointer** is a variable whose value is the address of another variable.
- Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

```
type *var-name;
```

- Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable.
- Following are the valid pointer declaration:

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



- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address.

- The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Using Pointers in C++

- There are few important steps, which we will do with the pointers very frequently:
  - **a)** We define a pointer variable.
  - **b)** Assign the address of a variable to a pointer.
  - **c)** Finally access the value at the address available in the pointer variable.
- This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.
- Following example makes use of these operations:

```
#include <iostream>

using namespace std;

int main () {
int  var = 20;   // actual variable declaration.
int  *ip;        // pointer variable

ip = &var;   // store address of var in pointer variable

cout<< "Value of var variable: ";
cout<< var <<endl;

// print the address stored in ip pointer variable
cout<< "Address stored in ip variable: ";
cout<<ip<<endl;

// access the value at the address available in pointer
cout<< "Value of *ip variable: ";
cout<< *ip<<endl;
```

```
   return 0;
}
```

- When the above code is compiled and executed, it produces result something as follows:

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Some concepts about pointers in C++

- Pointers have many but easy concepts and they are very important to C++ programming.
- There are following few important pointer concepts which should be clear to a C++ programmer

No.	Concept and Description
1.	**Null pointers** C++ supports null pointer, which is a constant with a value of zero defined in several standard libraries.
2.	**Pointer Arithmetic** There are four arithmetic operators that can be used on pointers: ++, --, +, -
3.	**Pointers vs. Arrays** There is a close relationship between pointers and arrays.
4.	**Array of Pointers** You can define arrays to hold a number of pointers.
5.	**Pointer to Pointer** C++ allows you to have pointer on a pointer and so on.

6.	**Passing pointers to functions** Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.
7.	**Return pointer from functions** C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.

Null Pointer

- It is always a good practice to assign the pointer **NULL** to a pointer variable in case you do not have exact address to be assigned.
- This is done at the time of variable declaration. A pointer that is assigned **NULL** is called a **null** pointer.
- The **NULL** pointer is a constant with a value of zero defined in several standard libraries, including iostream. Consider the following program:

```
#include <iostream>

using namespace std;
int main () {
int  *ptr = NULL;
cout<< "The value of ptr is " <<ptr ;

   return 0;
}
```

- When the above code is compiled and executed, it produces the following result:

```
The value of ptr is 0
```

- If a pointer contains the null (zero) value, it is assumed to point to nothing.
- To check for a null pointer you can use an if statement as follows:

```
if(ptr)     // succeeds if p is not null
if(!ptr)    // succeeds if p is null
```

- There are four arithmetic operators that can be used on pointers: ++, --, +, and -.
- To understand pointer arithmetic, let us consider that **ptr** is an integer pointer which points to the address 1000.
- Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer:

```
ptr++
```

- The **ptr** will point to the location 1004 because each time **ptr** is incremented, it will point to the next integer.
- This operation will move the pointer to next memory location without impacting actual value at the memory location.
- If **ptr** points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

- The following program increments the variable pointer to access each succeeding element of the array:

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
int  var[MAX] = {10, 100, 200};
int  *ptr;

// let us have array address in pointer.
ptr = var;

for (int i = 0; i< MAX; i++) {
cout<< "Address of var[" <<i<< "] = ";
cout<<ptr<<endl;

cout<< "Value of var[" <<i<< "] = ";
cout<< *ptr<<endl;

// point to the next location
ptr++;
   }
   return 0;
}
```

- When the above code is compiled and executed, it produces result something as follows:

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Decrementing a Pointer

- The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

```
#include <iostream>

using namespace std;
const int MAX = 3;

int main () {
int  var[MAX] = {10, 100, 200};
int  *ptr;

// let us have address of the last element in pointer.
ptr = &var[MAX-1];

for (int i = MAX; i> 0; i--) {
cout<< "Address of var[" <<i<< "] = ";
cout<<ptr<<endl;

cout<< "Value of var[" <<i<< "] = ";
cout<< *ptr<<endl;
```

```
// point to the previous location
ptr--;
   }
   return 0;
}
```

- When the above code is compiled and executed, it produces result something as follows:

```
Address of var[3] = 0xbfdb70f8
Value of var[3] = 200
Address of var[2] = 0xbfdb70f4
Value of var[2] = 100
Address of var[1] = 0xbfdb70f0
Value of var[1] = 10
```

Pointer Comparisons

- Pointers may be compared by using relational operators, such as ==, <, and >.
- If **p1** and **p2** point to variables that are related to each other, such as elements of the same array, then **p1** and **p2** can be meaningfully compared.
- The following program modifies the previous example one by incrementing the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1]

```
#include <iostream>
```

```
using namespace std;
const int MAX = 3;

int main () {
int  var[MAX] = {10, 100, 200};
int  *ptr;

// let us have address of the first element in
pointer.
ptr = var;
int i = 0;

while ( ptr<= &var[MAX - 1] ) {
cout<< "Address of var[" <<i<< "] = ";
cout<<ptr<<endl;
cout<< "Value of var[" <<i<< "] = ";
cout<< *ptr<<endl;
// point to the previous location
ptr++;
i++;
   }
   return 0;
}
```

- When the above code is compiled and executed, it produces result something as follows:

```
Address of var[0] = 0xbfce42d0
Value of var[0] = 10
Address of var[1] = 0xbfce42d4
```

```
Value of var[1] = 100
Address of var[2] = 0xbfce42d8
Value of var[2] = 200
```

Passing Pointers to Functions in C++

- C++ allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.
- Following a simple example where we pass integer pointer and an integer variable to a function and change the value inside the function which reflects back in the calling function:

```
#include <iostream>
using namespace std;
void salarychange(int  *var, int b)
{
    *var = *var+b;
}
int main()
{
int salary=0, bonus=0;
cout<<"Enter the employee current salary:";
cin>>salary;
cout<<"Enter bonus:";
cin>>bonus;
salarychange(&salary, bonus);
cout<<"Final salary: "<< salary;
    return 0;
}
```

- When the above code is compiled and executed, it produces the following result:

```
Enter the employee current salary: 100
Enter bonus: 20
Final salary: 120
```

- The function which can accept a pointer, can also accept an array as shown in the following example:

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

// function declaration:
double getAverage(int *arr, int size);

int main () {
   // an int array with 5 elements.
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   // pass pointer to the array as an argument.
   avg = getAverage( balance, 5 ) ;

   // output the returned value
cout<< "Average value is: " << avg <<endl;

   return 0;
}
```

```
double getAverage(int *arr, int size) {
   int i, sum = 0;
   double avg;

   for (i = 0; i< size; ++i) {
      sum += arr[i];
   }
   avg = double(sum) / size;

   return avg;
}
```

- When the above code is compiled and executed, it produces the following result:

```
Average value is: 214.4
```