Problem Solving and Programming 2

Lecture Four

More about Pointers in C++

Passing Pointers to Functions in C++

- A pointer can be passed to a function, just like any other argument is passed.
- A function in C can be called in two ways
  - Call by value
  - Call by reference
- To call a function by reference, you need to define it to receive the pointer to a variable in the calling function.
- Here is the syntax that you would use to call a **function by reference:**

```
type function_name(type *var1, type *var2, ...)
```

Advantages of Passing Pointers to Functions

- Passing a pointer to a function has two advantages:
  a) It overcomes the limitation of pass by value. Changes to the value inside the called function are done directly at the address stored in the pointer.
  b) Secondly, more than one value can be returned if we return multiple pointers or a pointer of an array.

Example of Passing Pointers to Functions

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

```
int add(int *x, int *y){
   int z = *x + *y;
   return z;
}
int main(){
   int a = 10, b = 20;
   int c = add(&a, &b);
   cout<<"The result = "<< c;

   return 0;
}
```

Passing an Array Pointer to a Function

- In C++ programming, the name of an array acts the address of the first element of the array; in other words, it becomes a pointer to the array.

```
#include<iostream>
#include<cmath>

using namespace std;
 int arrfunction(int x, float *arr){
   arr[0] = pow(x,2);
   arr[1] = pow(x, 3);
   arr[2] = pow(x, 0.5);
}

int main(){
   int x = 25;
   float arr[3];

   arrfunction(x, arr);

   cout<<"Square of "<< x<<" =\t"<<arr[0]<<endl;
   cout<<"Cube of " <<x<<" =\t" <<arr[1]<<endl;
   cout<<"Square root of "<< x<<" =\t"<< arr[2];

   return 0;
}
```

- When you run this code, it will produce the following output:

```
Square of 25 =   625
Cube of 25   =  15625
Square root of 25  =  5
```

Passing String Pointers to a Function

- A function can receive a pointer to characters similar to receiving a pointer to array.

Example

In this program, two strings are passed to the **compare**() function. A string here is an array of char data type.

- **strlen**() function can be used to find the length of the string.

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

```
void compare (char *x, char *y){
   if (strlen(x) > strlen(y)){
      cout<<"Length of Str1 is greater than or
equal to the length of Str2";
   }
   else{
      cout<<"Length of Str1 is less than the
length of Str2";
   }
}
int main(){

   char str1[] = "BAT";
   char str2[] = "BALL";
   compare(str1, str2);

   return 0;
}
```

- When you run this code, it will produce the following output:

```
Length of Str1 is less than the length of Str2
```

Return a Pointer from a Function

- In C++ programming, a function can be defined to have more than one argument, but it can return only one expression to the calling function.
- A function can return a single value that may be any type of variable, either of a primary type (such as **`int`**, **`float`**, **`char`**,

etc.), a pointer to a variable of primary or user–defined type, or a pointer to any variables.

Return a Static Array from a Function in C

- A function can also return an array using pointers. This can be done by returning the address of the first element in the array.

Example 1

- The following example shows how you can use a static array inside the called function (**`arrfunction`**) and return its pointer back to the **`main`**() function.

```
#include<iostream>

#include<math.h>

using namespace std;


float *arrfunction(int x){

   static float arr[3];

   arr[0] = pow(x, 2);

   arr[1] = pow(x, 3);

   arr[2] = pow(x, 0.5);

   return arr;  }


int main(){
   int x = 25;
   float *arr = arrfunction(x);
   cout<<"Square of "<< x <<" "<< *arr<<endl;
   cout<<"Cube of "<< x <<" "<<arr[1]<<endl;
   cout<<"Square root of "<< x <<" "<< arr[2];
   return 0;
}
```

Example 2

- The following function generates 10 random numbers.
- They are stored in a static array and return their pointer to the **main**() function.
- The array is then traversed in the **main**() function as follows

```
#include<iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;
/* function to generate and return random numbers */
int *getRandom() {
   static int  r[10];
   srand((unsigned)time(NULL));    /* set the seed */
   for(int i = 0; i < 10; i++){
      r[i] = rand();
   }
   return r;
}
int main(){
   int *p;     /* a pointer to an int */
   p = getRandom();
   for(int i = 0; i < 10; i++) {
      cout<< *(p + i)<<endl;
   }
   return 0;
}
```

Character Pointers and Functions in C

- A character pointer stores the address of a character type or address of the first character of a character array (`string`).
- Character pointers are very useful when working to manipulate the strings.
- An array of "`char`" type is considered as a string. Hence, a pointer of a char type array represents a string.

Declaring a Character Pointer

- A character pointer points to a character or a character array. Thus, to declare a character pointer, use the following syntax:
  `char *pointer_name;`

Initializing a Character Pointer

- After declaring a character pointer, you need to initialize it with the address of a character variable.
- If there is a character array, you can simply initialize the character pointer by providing the name of the character array or the address of the first elements of it.

Character Pointer of Character

- The following is the syntax to initialize a character pointer of a character type:

```
char *pointer_name = &char_variable;
```

Character Pointer of Character Array

- The following is the syntax to initialize a character pointer of a character array (string):

```
char *pointer_name = char_array;
/*or*/
char *pointer_name = &char_array[0];
```

Character Pointer Example

- In the following example, there are two variables character and character array.
- The two pointer variables are used to store the addresses of the character and character array, and then print the values of the variables using the character pointers.

```
#include<iostream>
```

```
using namespace std;
int main() {
  // Declare two variables
  char x = 'P';
  char arr[] = "C++ programming";

  // Declaring character pointers
  char *ptr_x = &x;
  char *ptr_arr = arr;

  // Printing values
  cout<<"Value of x : "<< *ptr_x<<endl;
  cout<<"Value of arr:  "<< ptr_arr;

  return 0;
}
```

- Run the code and check its output:

```
Value of x : P
Value of arr: C++ programming
```

Accessing Character Array

- To access each character of the character array, an asterisk (*) can be used before the character pointer name and then increment it.

Example

- Here is the full program code:

```
#include<iostream>

using namespace std;

int main(){
   char arr[] = "Character Pointers in C++";
   char *ptr = arr;

   while(*ptr != '\0'){
      cout<<*ptr;
      ptr++;
   }
}
```

- Run the code and check its output:

```
Character Pointers in C++
```

Character Pointers in C++

Example

- Alternatively, pass **`ptr`** to **`cout`** to print the string.

```
#include<iostream>
using namespace std;
int main(){
   char arr[] = "Character Pointers in C++";
   char *ptr = arr;

   cout<< ptr;
}
```

- On running this code, you will get the same output:

```
Character Pointers in C++
```