DHA SUFFA UNIVERSITY
Department of Computer Science
Object Oriented Programming
CS-1002L
Spring 2020

Lab 02

Pointers, References, Arrays and Functions

Objectives

- To learn about pointers, types of pointers, dynamic memory allocation, references and storing multiple values in Arrays, and working with functions.

Pointers

Pointer is a variable that holds the address of another variable.Pointer's name indicates ownership of the referred data.

Syntax

```
//Syntax of declaring pointer
dataType *name = address;

//Declaration of pointer variable in c++11
int *ptr = nullptr;
```

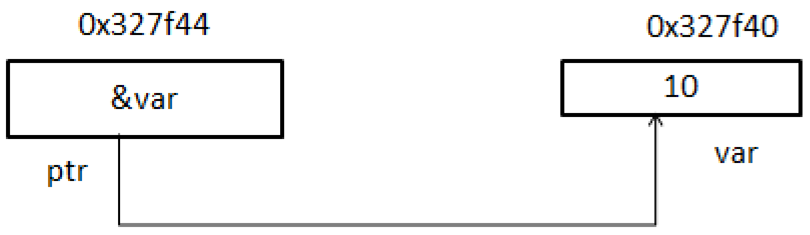


Fig 2.1: Pointer variable

There are four types of pointers:

1. Non constant pointer to non constant data
2. Non constant pointer to constant data
3. Constant pointer to non constant data
4. Constant pointer to constant data

Non constant pointer to non constant data
By using this type of pointer you can modify data by using dereference pointer and the pointer can be modified to point to other data

Syntax

```
//Syntax of declaring a non constant pointer to non constant data
dataType *variableName=NULL;

//Syntax of assigning a non constant pointer to non constant data
variableName=&variableName;
```

Example 2.1

```
//Example of non constant pointer to non constant data
#include <iostream>

int main ()
{
	//Variable Declaration
	int n1{7},n2{8},tmp{0};

	//Non constant pointer Declaration and initialization
	int *e1=&n1;
	int *e2=&n2;
	std::cout<<"========================\n";

	std::cout<<"\n The value before swapping are :\n";
	std::cout<<" Element1 = "<<*e1<<"\n Element 2= "<<*e2;

	//Implementing swapping algorithm
	tmp=*e2;
	*e2=*e1;
	*e1=tmp;

	//printing values after swapping
	std::cout<<"\n The value after swapping are :\n";
	std::cout<<" Element 1= "<<*e1<<"\n Element 2= " <<*e2;
	return 0;
}
```

Output

```
========================

 The value before swapping are :
 Element1 = 7
 Element 2= 8
 The value after swapping are :
 Element 1= 8
 Element 2= 7
```

Non constant pointer to constant data

By using this type of pointer we can modify a pointer to point to any type of data while data cannot be modified through the pointer variable

Syntax

```
//Syntax of non constant pointer to constant data
const dataType *variableName=&variableName;
```

Example 2.2

```
//Example of non constant pointer to constant data
#include <iostream>

int main ()
{
    //Declaring and initializing variable
    int num{10};

    //Declaring and initializing constant pointer variable
    const int *ptr=&num;
    std::cout<<"Value of Number = "<<*ptr;

    //Assigning new value through pointer variable
    //This Line will generate error as we are trying to modify value of a
const variable
    *ptr=20;
    std::cout<<"\nValue of Number = "<<*ptr;
    return 0;
}
```

Output

```
In function 'int main()':
15:9: error: assignment of read-only location '* ptr'
```

Constant pointer to Non constant data

By using this type of pointer we cannot modify a pointer to point to any type of data It will always point to same memory location while data at that location can be modified through the pointer variable

Example 2.3

```
//Example of constant pointer to non constant data
#include <iostream>

int main ()
{
     //Declaring and initializing variable
     int num{10};

     //Declaring and initializing constant pointer variable
     int *const ptr=&num;

     //Printing value of pointer variable
     std::cout<<"Value of Number = "<<*ptr;

     //Assigning new value through a pointer variable
     *ptr=20;
     std::cout<<"\nValue of Number = "<<*ptr;
     return 0;
}
```

Output

```
Value of Number = 10
Value of Number = 20
```

Constant pointer to constant data

By using this type of pointer we cannot modify a pointer to point to any type of data It will always point to same memory location while data at that location also cannot be modified through the pointer variable

Example 2.4

```
//Example of constant pointer to constant data
#include <iostream>

int main (){
    //Declaring and initializing variable
    int num{10};

    //Declaring and initializing constant pointer variable
    const int *const ptr=&num;
    std::cout<<"Value of Number = "<<*ptr;
    return 0;
}
```

Output

```
Value of Number = 10
```

Memory Allocation

Memory Allocation means to reserve space for a variable in memory.There are two types of memory allocation:

- Compile time memory allocation
- Runtime memory allocation

• **Compile time allocation of memory**: where the memory for named variables is allocated by the compiler. Exact size and storage must be known at compile time and for array declaration, the size has to be constant.

• **Runtime allocation or Dynamic allocation of memory:** where the memory is allocated at runtime and the allocation of memory space is done dynamically within the program run and the memory segment is known as a heap or the free store. In this case, the exact space or number of the item does not have to be known by the compiler in advance. Pointers play a major role in this case. To allocate space dynamically, use the operator new, followed by the type being allocated.To deallocate space dynamically, use the operator delete keyword

`new` Operator

`new` operator id used to dynamically allocate memory

Syntax

```
//Syntax of new operator
Pointer Variable=new data-type(value)
int *p=new int(10);
```

`delete` Operator

`delete` Operator is used to deallocate memory

Syntax

```
// Syntax of delete Operator
delete pointer-variable;
delete p;
```

Example 2.5

```
//Example of dynamic memory allocation
//Example:main.cpp
#include <iostream>

int main()
{
   int *ptr=nullptr;
   //Declaring a memory with value 20 and assigning its address to
ptr
   ptr = new int(20);
   std::cout<<"Value of ptr = "<<ptr<<std::endl;
   std::cout<<"Value at address pointed to by ptr =
"<<*ptr<<std::endl;

   //deallocating all the memory created by new operator
   delete ptr;
   std::cout<<"Value of ptr = "<<ptr<<std::endl;
   std::cout<<"Value at address pointed to by ptr =
"<<*ptr<<std::endl;
   return 0;
}
```

Output

```
Value of ptr = 0x2a376b0
Value at address pointed to by ptr = 20
Value of ptr = 0x2a376b0
Value at address pointed to by ptr = 0
```

References

When a variable is declared as reference, it becomes an alternative name for an existing variable. A variable can be declared as reference by putting '&' in the declaration. Three major differences between references and pointers are

1. You cannot have NULL references.
2. Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
3. A reference must be initialized when it is created. Pointers can be initialized at any time.

Example 2.6

```
#include <iostream>

int main()
{
  //declaring simple variable
  int i{0};
  double d{0} ;

  //declaring references variable
  int& r = i;
  double& s = d;
  i = 5;
  std::cout << "Value of i:"<< i << std::endl;
  std::cout << "Value of reference:"<< r << std::endl;
  d=11.7;
  std::cout << "Value of d:"<< d << std::endl;
  std::cout << "Value of d reference:"<< s << std::endl;
}
```

Output

```
Value of i:5
Value of reference:5
Value of d:11.7
Value of d reference:11.7
```

Arrays

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

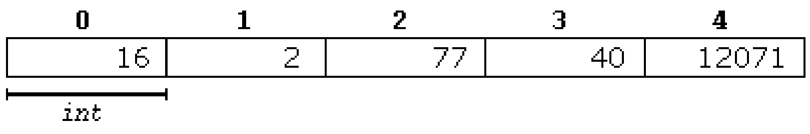
Instead of declaring individual variables, such as number0 , number1 , ..., and number99 , you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

Initializing Arrays

Syntax

```
int my_array[5] = { 16, 2, 77, 40, 12071 };
```

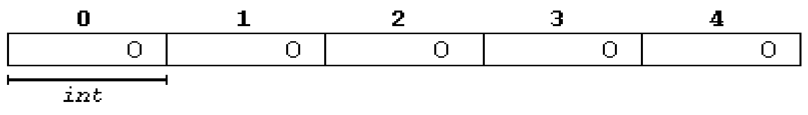
Above syntax declares an array that can be represented like this:



Syntax

```
int my_array [5] = { };
```

Above syntax declares an array that can be represented like this:



When an initialization of values is provided for an array, C++ allows the possibility of leaving the square brackets empty `[]`. In this case, the compiler will assume automatically a size for the array that matches the number of values included between the braces `{}`. Example is shown below.

Syntax

```
int my_array [] = { 16, 2, 77, 40, 12071 };
```

After this declaration as above, array foo would be 5 int long, since we have provided 5 initialization values.

Accessing the values of Array

Syntax

```
//the following statement stores the value 75 in the third element of array:

my_array [2] = 75;

//the following statement copies the value of the third element of array to
a variable called x

x = my_array[2];
```

Example 2.7

```
// arrays example
// Program to sum elements of array
#include <iostream>

int main ()
{
  //initialization

  int my_array[] = {16, 2, 77, 40, 12071};
  int n {0}, result {0};

  //sum of array elements
  for ( n=0 ; n<5 ; ++n )
  {
    result += my_array[n];
  }
  //printing on screen
  std::cout << result;

  return 0;
}
```

Output

```
12206
```

Dynamically Allocating Arrays

In addition to dynamically allocating single values, we can also dynamically allocate arrays of variables. Unlike a fixed array, where the array size must be fixed at compile time, dynamically allocating an array allows us to choose an array length at runtime. To allocate an array dynamically, we use the array form of new and delete (often called `new[ ]` and `delete[]`).

Example 2.8

```
#include <iostream>

int main()
{
    std::cout << "Enter a positive integer: ";
    int length;
    std::cin >> length;

    // use array new.  Note that length does not need to be constant!
    int *array = new int[length];

    std::cout <<"The allocated length of array is " << length << '\n';

    // set element 1 to value 6
    array[1] = 6;

    //Print first element of array
    std::cout << "Value at index 1 : "  << array[1];

    // use delete [] to deallocate array
    delete[] array;
    return 0;
}
```

Output

```
Enter a positive integer: 3
The allocated length of array is 3
Value at index 1 : 6
```

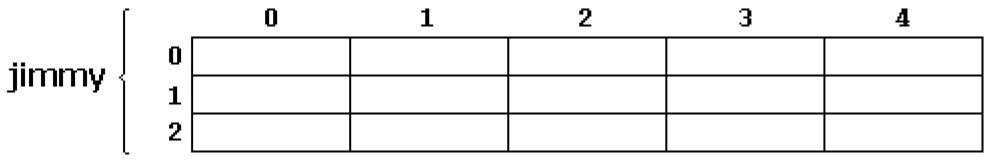
Multi Dimensional Array

Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a two-dimensional table made of elements, all of them of a same uniform data type.

Syntax

```
int jimmy [3][5];
```

The above syntax will create an array like shown below.



Example 2.9

```
#include <iostream>
//Printing an array of 3 by 3 in tabular form
int main ()
{
   // an array with 3 rows and 3 columns.
   int a[3][3] = { {0,0,0}, {1,2,3}, {2,4,0}};
   // output each array element's value
   for ( int i = 0; i < 3; ++i)
   {
      for ( int j = 0; j < 3; ++j )
      {
         std::cout << a[i][j]<< "\t";
      }
      std::cout << "\n" ;
   }
   return 0;
}
```

Output

```
0      0      0
1      2      3
2      4      0
```

Functions

A function groups a number of program statements into a unit and gives it a name. This unit can be invoked from other parts of the program.

The three necessary components required to add a function to a program are:
1.The function declaration or function prototype.
2.The call to the function.
3. The function definition or function body.

The general structure of a function declaration is as follows:

```
return_type function_name(arguments);
```

The general structure of a function definition is as follows:

```
return_type function_name(parameter list)
{
   body of the function
}
```

There are four types of functions depending on the return type and arguments:

- Functions that take no-arguments and return nothing.

- Functions that take arguments but return nothing.
- Functions that take no-arguments but return something.
- Functions that take arguments and return something.

Consider an example that calculates the product of two numbers sum(n1,n2) .

Example 2.10

```
#include <iostream>
//declaring product function
int product (int n1,int n2);

//Defining product function
int product(int n1,int n2)
{
    //Declaring variables
    int result {0};
    //Calculating product
    result = n1 * n2;
    //returning result
    return result;
}
//Defining main function
int main()
{
    //Declaring variables
    int num1 {0} , num2 {0},prod {0};
    std::cout << "Enter Number 1\n";
    //Taking input
    std::cin >> num1;
    std::cout << "Enter Number 2\n";
    //Taking input
    std::cin >> num2;
    //calling product function
    prod = product(num1,num2);
    std::cout << "Product of two Numbers:\n" << prod;
    return 0;
}
```

Output

```
Enter Number 1
3
Enter Number 2
9
Product of two Numbers:
27
```

There are three ways to pass value or data to function in C++ language:

1. Pass by value.
2. Pass by reference.
3. Pass by pointer reference.

Pass by Value in C++

In call by value, original value is not modified.In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of the function parameter, it is changed for the current function only. It will not change the value of the variable inside the caller method such as main().

Example 2.11

```
#include <iostream>
// function declaration
void swap(int n1, int n2);

// function definition
void swap(int n1, int n2)
{
   int temp=0;
   temp = n1;
   n1 = n2;
   n1 = temp;
}

int main ()
{
   // local variable declaration:
   int num1 {100},num2{200};
   std::cout << "Before swap, value of num1 :" << num1 << "\n";
   std::cout << "Before swap, value of num2 :" << num2 << "\n";

   /* call by value.*/
   swap(num1, num2);

   std::cout << "After swap, value of num1 :" << num1 << "\n";
   std::cout << "After swap, value of num2 :" << num2 << "\n";

   return 0;
}
```

Output

```
Before swap, value of num1 :100
Before swap, value of num2 :200
After swap, value of num1 :100
After swap, value of num2 :200
```

Pass by Reference

In pass by reference, original value is modified because we pass reference (address).Here, the address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Below is an example to show how we can pass arguments in function by references of variables.

Example 2.12

```
#include <iostream>

// function declaration

void swap(int &num1, int &num2);

// function definition

void swap(int &num1, int &num2)

{

   int temp=0;

   temp = num1;

   num1 = num2;

   num2 = temp;

}

int main ()

{

   // local variable declaration:

   int num1 {100},num2 {200};

   std::cout << "Before swap, value of num1 :" << num1 << "\n";

   std::cout << "Before swap, value of num2 :" << num2 << "\n";

   /* calling a function to swap the values using variable reference.*/
   swap(num1, num2);

   std::cout << "After swap, value of num1 :" << num1 << "\n";

   std::cout << "After swap, value of num2 :" << num2 << "\n";

   return 0;

}
```

Output

```
Before swap, value of num1 :100
Before swap, value of num2 :200
After swap, value of num1 :200
After swap, value of num2 :100
```

Pass by pointer reference

The call by pointer method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

To pass the value by pointer, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments.

Example 2.13

```
#include <iostream>

// function declaration

void swap(int *num1, int *num2);

// function definition

void swap(int *num1, int *num2)

{
   int temp=0;

   temp = *num1;

   *num1 = *num2;

   *num2 = temp;

}

int main ()

{

   // local variable declaration:

   int num1 {100},num2 {200};

   std::cout << "Before swap, value of num1 :" << num1 << "\n";

   std::cout << "Before swap, value of num2 :" << num2 << "\n";

   /* calling a function to swap the values using pointer reference.*/

   swap(&num1, &num2);

   std::cout << "After swap, value of num1 :" << num1 << "\n";

   std::cout << "After swap, value of num2 :" << num2 << "\n";

   return 0;

}
```

Output

```
Before swap, value of num1 :100
Before swap, value of num2 :200
After swap, value of num1 :200
After swap, value of num2 :100
```

Lab Assignment

Task 2.1

1. Create a pointer `non-const pointer to non-const int.`
2. Update the value of the variable using the `non-const pointer.`
3. Print the value of the variable using `non-const pointer.`
4. Modify the value of `const` variable `i` through `i_ptr`. (You should expect errors here, write the reason for the error in comments using **multi-line** comments).

Task 2.2

1. Create a constant integer variable `var1` with initial value of 10 and a `const` reference to it as `ref_var1`.
2. Also, modify the value of `var1` with an `integer` value 2 using `ref_var1`.(You should expect an error here, write the reason for the error in comments using **multi-line** comments).
3. Create a non-constant integer variable **var2** with initial value 0..
4. Take input from User in **var2**.
5. Create `ref_var2` and refer to the **var2.**
6. Update **var2** using **ref_var2**.
7. Print both **var1** and **var2** using their references.

Task 2.3

Write a program to create a 2D array of 3 rows and 2 columns, ask the user for an input value for every index print it in a table format. [Hint:Assign values using nested for loop]. Also, use functions to print this array.

Task 2.4

Write a simple function accepts a character array as parameter and performs following:

1. Prints array.
2. Prints array in reverse order.

Task 2.5

Write a c++ program that performs multiplication of all array (2D array) elements and displays result. (Use methods for implementing this task).

Also,

Consider,

$$\text{Matrix A} = \begin{matrix} 1 & 4 \\ 2 & 6 \\ 4 & 9 \end{matrix}$$

$$\text{Matrix B} = \begin{matrix} 4 & 5 & 4 \\ 2 & 10 & 0 \end{matrix}$$

Write a C++ program two add these matrices as Matrix C, display Matrix C on console in the same format.

Task 2.6

Consider you have an integer array A={1,2,5,7,5} of size 5. Now,Create following functions in C++ to:

1. return first element of array. // returns 1
2. return last element of array. // returns 5
3. return maximum number in array. // returns 7
4. return minimum number in array. // returns 1
5. assign a specific value to all the array elements. [Hint: takes two arguments, arr and number to assign]. // function(2,arr) -> updated array = {2,2,2,2,2}
6. Find the given element in an array and returns its index, -1 if number does not exist, 1st index if number is repeating in an array. // function(7, arr) -> returns 3

Submission Instructions

- Name your Task Files as T1,T2, etc (example : T1.cpp)
- Name your home assignment as Lab_0X .cpp ()
- Create a new folder named cs152abc where abc is your 3 digit roll #. e.g. cs152111.
- Copy all the task and assignment files into this folder.
- Now make a zip file of above made folder named cs152abc.zip is created e.g. cs152111.zip
- Upload the this zipped folder on LMS under the assignment named Lab 01_Assignment – XX, where XX is your section name.