Q.8 Write a C++ program to read the address of a person and exit when the user enters dot (.) from the keyboard.

Q.9 Write a C++ program to find out the area of a triangle and if any side is zero then display the message "There is no triangle".

Q.10 Write a C++ program to input a character from the keyboard and display the message after testing whether it is Vowel or Consonant.

UNIT 5

ARRAYS AND STRINGS

After the completion of Unit -5, Students will be able to:

- Explain the concept of an array
- Know how array elements are arranged in memory
- Explain the following array terminology
  - Name
  - Size
  - Index
- Explain how to define and initialize an array of different size and data types
- Explain how to access and write at an index in an array
- Explain how to traverse an array using all loop structures
- Use the **sizeof()** operator to find the size of an array
- Explain two dimensional array
- Explain how to define and initialize a two dimensional array of different size and data types
- Explain how to access and write at an index in a two dimensional array
- Explain strings
- Explain how to define and initialize a string
- Explain the most commonly used string functions

INTRODUCTION

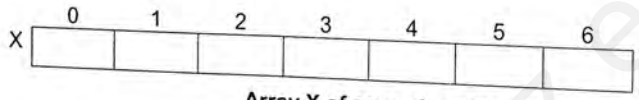
This unit is about arrays i.e. its definition, representation, accessing and traversing. There are two types of array data structures, one dimensional and two dimensional. The unit also focuses on strings and its most commonly used functions.

5.1 Introduction to array

We group our everyday data into conceptual units rather than storing in individual pieces. We collect the data items into conceptual units and assign names to them. For a single conceptual unit we have a single name. This conceptual unit of homogeneous data having the same name for all the data items is called array. The array is used to store many items of the same type as a single unit or group. Consider the examples of storing the scores of all the students of a class in an array or storing their names.

5.1.1 Concept of arrays

An array is a collection of data items of the same data types placed in contiguous memory locations that can individually be referenced by using an index to a unique identifier. Consider the example for storing 7 values of type *int* in an array without having to declare 7 different variables each one with a different variable name. Instead of that, using an array we can store 7 different values of the same data type with a unique variable name. Consider an array X that contains 7 integer values of type *int*. It can be represented as:



Array X of seven locations

These elements are numbered from 0 to 6 since in arrays the first index is always 0.

5.1.2 Representation of array

Array elements are stored in consecutive memory locations inside the computer memory. For example, if in the above array X we store the marks of seven students and the first element X[0] is stored at address 100, inside the computer memory, then X[1], X[2], X[3], X[4], X[5], X[6] will be stored at memory addresses 102,104,106, 108, 110, and 112 respectively. Here, the difference in the values of the memory addresses is two numbers because *an* ***int*** *data type occupies* 2 bytes of memory space. It is shown in figure 5.1.

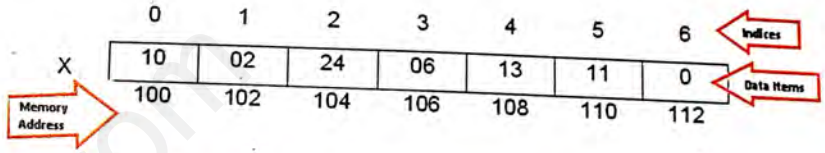


Figure 5.1: Memory Representation of 1-D Array for an int Data Type

Consider another example, shown in figure 5.2, of one dimensional array named SUBJ for storing the title of a subject whose length is 7 characters, such as, PHYSICS.

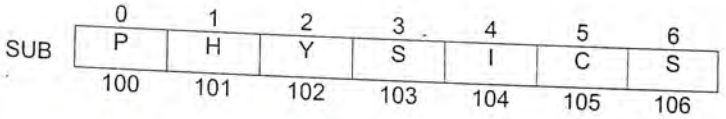


Figure 5.2: Memory Representation of 1-D Array for a char Data Type

5.1.3 Terminology used in Arrays

Arrays can be understood well by first knowing some commonly used terminologies such as name, size, and index.

a. Name of an array

The name of the array can be any valid identifier. Each location of the array has the same name with different index. In the above example, all the 7 locations of the array X has the same name X but different indices, X[0], X[1], X[2], X[3], X[4], X[5], and X[6].

b. Size of the array

The number of elements that can be stored in an array is called the size or length of that array.

For example:

float Z [10];

Here, size of the array Z is 10, which means that we can store 10 floating point values in this array.

c. Index

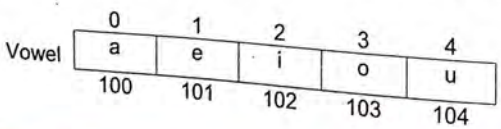
The location number of an item in an *array* is called index or subscript of that element. For example 0,1,2,3,4,5 and 6 are the indices of the elements of the array int X[7].

5.1.4 Definition and initialization of an array

As we have two types of arrays, one dimensional and two dimensional, both have their own ways of definition. Here, we are defining and initializing one dimensional array.

a. One dimensional array

A one-dimensional array or linear array can be represented as a single row of contiguous memory locations. Accessing of its elements involves a single subscript.



Like a regular variable, an array must be declared before it is used. A typical declaration for a one dimensional array in C++ is:

data type name [***size***];

Here, the data type is a valid data type (such as int, float...), name is a valid identifier and enclosed in square brackets [ ] is the size of the array which means that how many elements this array can accommodate.

For Example:

int x[7];
char vowel[5];
float temperature[7];

b. Initializing one dimensional array

Arrays can be initialized, like variables, when created. C++ provides a convenient way to initialize entire array via use of an initializer list.

The following example shows the initialization of an *integer* array.

```
// initializing array using initializer list method
#include <iostream.h>
#include <conio.h>
int main()
{
int A[5] = { 3, 2, 7, 5, 8 };
cout << A[0] << endl;
cout << A [1] << endl;
cout << A [2] << endl;
cout << A [3] << endl;
```

```
cout << A [4] << endl;
getch();
return 0;
}
```

Output of the program

3
2
7
5
8

If we do not initialize all of the elements in an array then the remaining elements will be initialized to 0.

Similarly, to initialize all the elements of an array to 0, we can define that array as:

```
int A[5]={0};
```

5.1.5 Accessing and writing at an index in an array

Accessing of array means to get into an index in an array and take its value for some operation while by writing at an index of an array is storing some value at a particular index of the array. These are the two important actions that should be taken while dealing with arrays.

The following section addresses them with examples.

Accessing an array

Array elements are accessed by using respective indices. The format for accessing is as follows:

Array name[index];

Looking at the example discussed in the above section, in which A has 5 elements, each of the element has been accessed as A[0], A[1], ..., A[4] using cout statement.

To access all elements of an array we use loops instead of individual reference to each element, For Example:

```
// accessing array using loop
#include <iostream.h>
#include <conio.h>
int main()
{
int A[5] = { 3, 2, 7, 5, 8 };
for(int i=0; i<=4; ++i)
cout << A[i] << endl;
getch();
return 0;
}
```

Output of the program

3
2
7
5
8

Here, **for** loop has been used to access the elements of the array one by one and print the result on the screen.

Writing at an index in an array

Accessing of an index of an array is followed by the writing operation, only in the case, if we want to store a new value at that index or to update the existing one. For example, if we want to add the values in the array **A**, we can individually access each array location and write the value using assignment operator or using input statement (cin) It is shown in the following program.

```
// writing at in index of an array
#include <iostream.h>
#include <conio.h>
int main()
{
int A[5];
A[0]=12;
A[1]=-20
cout<< "Enter number"<<endl;
cin >> A[2];
cout<< "Enter number"<<endl;
cin >> A[3];
cout<< "Enter number"<<endl;
cin >> A[4];
getch();
return 0;
}
```

Output of the program

```
Enter number
7
Enter number
5
Enter number
8
```

Loops can also be used for writing at all indexes of an array. Consider the following example having an array *Temperature* that accepts the values of weekly temperature from the users at run time and write them into their respective indexes.

```
// writing an array using loop
#include <iostream.h>
#include <conio.h>
int main()
{
float Temperature[7];
cout<< "Enter 7 days temperature:"<<endl;
for(int i=0; i<7; ++i)
cin >>Temperature[i]; // writing at indexes
cout<< "The last week temperature was:"<<endl;
for(int i=0; i<7; ++i)
cout<<Temperature[i] << ", "; // accessing the elements
getch();
return 0;
}
```

Output of the program

```
Enter 7 days temperature:
35.9 22.0 27.3 25.5 28.3 31.2 33.7
```

The last week temperature was:

35.9, 22.0, 27.3, 25.5, 28.3, 31.2, 33.7

5.1.6 Traversing an array

Accessing the elements of an array is called traversing of array.

```
// reading numbers into an array and then searching for an item
#include <iostream.h>
#include <conio.h>
int main()
{
int a[10]; // Declare an array of 10 integers
int n = 0;
cout<< "Enter 10 integer values"<<endl;
while (n<10)
{
cin >> a[n];
n++;
}
int item; // declare an item to search
cout<< "Enter an item to search"<<endl;
cin>>item;
for (int i=0; i<10; i++) //traverse array
{
if(item!=a[i]) // comparison
continue;
else
{
cout<< "Item "<<item<<" is found at location "<<i+1;
goto end;
}
}
cout << "Search unsuccessful: "<<item<< " is not found";
end:
getch();
return 0;
}
```

Consider another program of array traversal using *for* loop to display those values from the array which are odd.

```
#include <iostream.h>
#include <conio.h>
int main()
{
int A [] = {3, 2, 7, 5, 8};
for ( int i=0 ; i<5 ; i++ )
{
if (A[i] % 2 !=0)
cout<< A[i] << endl;
}
getch();
return 0;
}
```

Output of the program

```
3
7
5
```

5.1.7 sizeof() Operator

The **sizeof()** operator can be used with arrays to return the size allocated for the entire array. Consider the following program:

```
// size of an array example
#include <iostream.h>
#include <conio.h>
int main()
{
int A[] = { 0, 1, 2, 3, 4 };
cout << "Size of the array A is: " <<sizeof (A); // 5 elements * 2 bytes for each element
getch();
return 0;
}
```

Output of the program

```
Size of the array A is: 10
```

Using the **sizeof()** operator, we can also find out the number of elements in an array. Consider the following example:

```
/* to find the number of elements in array using
sizeof() operator*/
# include <iostream.h>
#include <conio.h>
int main()
{
int A[] = { 0, 1, 2, 3, 4 };
cout << "Size of the array ="<<sizeof(A)<<" bytes ";
int Total_Elements = sizeof(A) / sizeof(A[0]);
cout<<"\nTotal number of elements in the array A
are: "<<Total_Elements;
getch();
return 0;
}
```

Output of the program

```
Size of the array =10 bytes
Total number of elements in
the array A are: 05
```

5.2 Two-dimensional arrays

An array having more than one subscripts is called *multidimensional* array. A multidimensional array is an "array of arrays". It is not limited to two indices i.e., two dimensions, but can also have three-dimensions and four-dimensions etc.

5.2.1 Concept of two dimensional arrays

The most communally used multidimensional array is two dimensional arrays which is also called *bi-dimensional* array. It is used for storing of tabular data arranged in rows and columns. It can be represented by two indices i.e. row index and columns index.

All the elements of two dimensional array are of the same data type. Consider the following array:

```
int X[3][5];
```

Array X consisting of three rows and five columns. It can be represented as:

	0	1	2	3	4
0	12	-123	58	95	67
1	56	45	23	45	176
2	78	-56	90	55	74

Two dimensional array is also called *matrix* in mathematics and *table* in business applications.

5.2.2 Definition and initialization of two dimensional arrays

Like one-dimensional array, a two-dimensional array is defined and initialized before using it in a program. The following sections describe its definition and initialization with the help of examples.

a. Defining two dimensional array

The general syntax for the declaration of two dimensional arrays is given below.

Data Type **Array name [row index] [column index];**

In the above syntax, the *data type* is any valid data type, such as *int, float, long etc.,* which is followed by array name and two indices. The first index is for number of rows and the second index is for number of columns in the array.

A *float* type array, **tdArray**, consisting of 2 rows and 2 columns can be defined by using the following statement.

```
float tdArray [2][2];
```


b. Initializing two dimensional arrays

To initialize a two-dimensional array, it is easiest to use nested braces, with each set of numbers representing a row:

```
// two dimensional array initialization
int anArray[3][5] =
{
{ 1, 2, 3, 4, 5 }, // row 0
{ 6, 7, 8, 9, 10 }, // row 1
{ 11, 12, 13, 14, 15 } // row 2
};
```

If we want to initialize a *two-dimensional array* to 0 then it can be done as follows:

```
int anArray[3][5] = { 0 };
```

5.2 .3 Accessing and writing at an index in a two dimensional array

To operate on the elements of two dimensional arrays, the elements should be first accessed and then updated. Writing at an index of an array is also done when a new two dimensional array is declared. Accessing and writing at an index of a two dimensional array is similar to one dimensional array. The following sections discuss them with the help of examples.

a. Accessing two dimensional arrays

Accessing two dimensional arrays need two indices, one for the row and the other for the column. For example, if we have an array anArray[3][5] and want to access an element at second row and third column. Then it can be written as:

```
anArray[1][2];
```

Accessing elements of a two-dimensional array individually is very difficult and time consuming. A better solution to this is the application of loop. Normally, nested loops are used for this purpose: outer loop for the row, and inner loop for the columns. Since, two-dimensional arrays are typically accessed row by row, therefore, the row index is used as the outer loop and the column index is used as the inner loop.

Consider the following program used to access the elements of the array, ***anArray[3][5]***, initialized in the above section and display them on the screen in a matrix form.

```
// accessing two dimensional array using loop
#include <iostream.h>
#include <conio.h>
int main()
{
int anArray[3][5] =
{{ 1, 2, 3, 4, 5 },
{ 6, 7, 8, 9, 10 },
{ 11, 12, 13, 14, 15 }};
for ( int row=0; row<3; ++row)
{
 for (int col=0; col<5; ++col)
 cout<<anArray [row][col]<< ", ";
 cout<<endl;
 }
 getch();
 return 0;
 }
```

Output of the program

1, 2, 3, 4, 5
6, 7, 8, 9, 10
11, 12, 13, 14, 15

b. Writing at an index of two dimensional array

Loop is a better way of writing to the indices of a two dimensional array. Consider the following example:

```
// writing in a two dimensional array using loop
#include <iostream.h>
#include <conio.h>
int main()
{
int A[2][3];
cout<< "Kindly enter values for the array"<<endl;
for (int i = 0; i < 2; i++)
for (int j = 0; j < 3; j++)
cin>> A[i][j]; // writing to the array
cout<< "Values of the array are:"<<endl;
for (int i = 0; i < 2; i++)
{
for (int j = 0; j < 3; j++)
cout << A[i][j]<< "\t"; //accessing of the array
cout<<endl;
}
getch();
return 0;
}
```

Output of the program

2 -1 7 1 0 10
Values of the array are:
2 -1 7
1 0 10

5.3 Strings

5.3.1 What is String?

The sequence of characters enclosed in quotation marks and is used to handle non-numeric data i.e. names, addresses etc. are called strings.

5.3.2 Defining a C String

General syntax for defining string in C is given below:

data type name_of_string [size of string];

In the above general syntax, *data type*, is **char** data type and *name_of_string* is the name of the string. The size (number of characters) of the string is specified in square brackets, [ ]. The following statement defines a string with the name *str* having size 20.

char str [20];

For handling strings, the header file <*cstring*> is needed but in some implementations this library may be automatically included when the header file <*iostream*> is included. There are some limitations of strings in C. To overcome and remove these problems, C++ provides a class <*string*> that removes many of these problems. This class provides some typical string operations like *comparison*, *concatenation*, *find* and *replace*, and a function for obtaining *substrings*. The general syntax for defining strings in C++ is given below:

string str;

This statement defines a string name *str* of type *string*.

5.3.3 Initializing string

1. *Strings in* **C** can be initialized as arrays are initialized because *string* is a character array.

Consider the following example:

char greeting [6] = { 'H', 'e', 'l', 'l', 'o', '\0' };

The above statement declares a string named "*greeting*" with six elements of type *char* that are initialized with the characters that form the word "Hello" plus a null character '\0' at the end. Another method for initializing a string in C is to enclose the string in double quotes, "" as shown below:

char greeting [ ] = "Hello";

In both cases, the array of characters **greeting** is declared with a size of 6 elements of type *char*: the 5 characters that compose the word "Hello" plus a final null character ('\0') which specifies the end of the sequence and that, in the second case, when using double quotes (") it is appended automatically.

Consider an example to ask for your name and then display a greeting message on the screen as shown below using string:

```
#include <iostream.h>
#include<conio.h>
int main()
{
char question[] = "Please, enter your first name: ";
char greeting[] = "Hello, ";
char yourname [80];
```

```
cout << question;
cin >> yourname;
cout << greeting << yourname << "!";
getch();
return 0;
}
```

Output of the program

```
Please, enter your first name:
Rahman
Hello, Rahman!
```

2. *Strings in* **C++** can be initialized both ways like in C and directly like ordinary variables with the only difference that the string should be enclosed in double quotes. Consider the following initialization examples:

```
string str1("Call me");
string str2 = "Send sms!";
string str3("OKay");
```

Here, in these examples three strings: *str1*, *str2* and *str3* are directly initialized to "Call me", "Send sms" and "OKay" respectively. Note that while using C++ string in programs, the class *string* must be included.

Consider the following program to demonstrate initialization of C++ string:

```
// C++ strings initialization program
#include <iostream.h>
#include<conio.h>
```



```
#include<string>
int main()
{
string str1("Call me");
string str2 = "Send SMS";
string str3("OKay");
cout<< "First string str1 is: "<<str1<<endl;
cout<< "Second string str2 is: "<<str2<<endl;
cout<< "Third string str3 is: "<<str3<<endl;
getch();
return 0;
}
```

Output of the program

```
First string str1 is: Call me
Second string str2 is: Send SMS
Third string str3 is: Okay
```

The following program demonstrates some more features of C++ string e.g. concatenation, copying and finding size of string.

```
#include <iostream.h>
#include<conio.h>
#include<string>
int main()
{
string str1="Hello";
string str2 = "World";
```

```
string str3, str4;
int len;
str3 = str1; // String copying
str4 = str1 + str2; // String concatenation
len = str4.size( ); // Finds string size
cout<<"str3=\t"<<str3<<endl;
cout<<"str4=\t"<<str4<<endl;
cout<<"length of str4 = \t"<<len<<endl;
getch();
}
```

Output of the program

```
str3=     Hello
str4=     HelloWorld
length of str4      = 10
```

5.3.4 Commonly used string functions

The header file *string* provides many string manipulation functions. These are discussed with the help of programs as follows.

a. Concatenation of strings

Strcat() function

Combining two strings into a single string is called string concatenation. *Strcat()* function is used for this purpose. The following program demonstrates the implementation of *strcat()* function for the concatenation of the two strings.

```
// strings concatenation program
#include <iostream.h>
```



```
#include <string.h>
#include<conio.h>
int main()
{
char FirstName[15] = "SSC ";
char LastName[15] = "Computer";
strcat(FirstName, LastName);
cout << FirstName;
getch();
return 0;
}
```

Output of the program

```
SSC Computer
```

b. Copying of strings

copy() function

copy() function is used to copy one string to another string. The general syntax of *copy* function is given below:

str.copy (string_name, number_of_characters, starting_pointer)

Here, the *string_name* is the name of the string to which the string is to be copied. *number_of_characters* represents the exacts number of characters to be copied and *starting_pointer* indicates the starting point from where to copy. Consider the following program:

```
// strings copying program using copy() function
#include <iostream.h>
```

```
#include <string>
#include<conio.h>
int main()
{
string str = "Welcome to C++ Strings";
char x[55];
cout << "str is: " << str << endl;
str.copy(x, 7, 0); // copy 7 characters of string str into string x, starting from position zero
cout << "The copied string is: " << x << endl;
getch();
return 0;
}
```

Output of the program

Welcome to C++ Strings
The copied string is: Welcome

strcpy() function

For copying string, another string function *strcpy()* can also be used. Consider the following program to demonstrate the use of *strcpy()*.

```
// copying string using strcpy() function
#include <iostream.h>
#include <string>
#include<conio.h>
main()
{
string SS;
char CC[17];
SS = "This is a string";
strcpy(CC,"This is a string");
cout << SS << endl; cout << CC << endl;
getch();
}
```

Output of the program

This is a string
This is a string

c. Finding a character or word in a strings

find() function

To find a particular word or a character in a string, *find ()* function is used. *find ()* function takes only one argument which is the desired character or word within the string. The following program demonstrates how *find()* function is used to search the word **interesting** and character **g** in the string "C++ is a very interesting programming language".

```
// finding a word or character using find() function
#include <iostream.h>
#include <string>
#include<conio.h>
main()
{
string str("C++ is a very interesting programming language");
int loc1, loc2;
loc1 = str.find ("interesting ");
cout << "Word interesting is found on position: " <<loc1+1;
loc2 = str.find('g');
cout << "\nFirst character 'g' found on position: "<< loc2;
```

```
getch();
return 0;
}
```

Output of the program

Word interesting is found on position: 15

First character 'g' found on position: 24

d. Finding length of a strings

length() function

To find the length of a string, *length* () function is used. Consider the following program to demonstrate working of the *length* () function.

```
#include <iostream.h>
#include <string>
#include<conio.h>
main()
{
string str("C++ is a very interesting programming language");
cout << "Length of the given string is: "<< str.length();
getch();
return 0;
}
```

Output of the program

Length of the given string is: 46

e. Swapping strings

swap() function

To swap (or interchange) values of two strings, *swap*() function is used. Consider the following program to demonstrate working of *swap* () function.

```
#include <iostream.h>
#include <string>
#include<conio.h>
int main()
{
string str1 = "F.Sc Part I";
string str2 = "F.Sc Part II";
cout << "str1 is: " << str1 << endl;
cout << "str2 is: " << str2 << endl;
str1.swap(str2);
cout << "str1 is: " << str1 << endl;
cout << "str2 is: " << str2 << endl;
getch();
return 0;
}
```

Output of the program

str1 is: F.Sc Part I
str2 is: F.Sc Part II
str1 is: F.Sc Part II
str2 is: F.Sc Part I

Summary

- A collection of homogeneous data items stored in consecutive memory locations is called array.
- An array can store data of any valid data type, e.g. integer, floating point and character. An integer array can only store integer data.
- There are two types of arrays: one dimensional and multi-dimensional. Two dimensional arrays is a type of multi-dimensional array.
- One dimensional array can be defined by using a single subscript.
- Two dimensional arrays can be represented with the help of two subscripts. It is also called matrix in mathematics and table in business application.
- One dimensional array can be defined by preceding the name of the array by a valid data type and followed by a single subscript.
- Two dimensional arrays can also be defined by preceding the name of the array by a valid data type, such as int, long and char etc., and followed by two subscripts.
- One dimensional and two dimensional arrays can be initialized in the same way during the declaration time.
- Arrays can be efficiently and easily processed with the help of looping structures.
- A sequence of characters enclosed in double quotes is called string.
- Different operations such as comparison, concatenation, copying, and finding length can be performed on strings.

Exercise

Q.1 Fill in the Blanks.

i. In C/C++ language, normally the indexing of an array starts from _____.
ii. The size of the array int x[10] is _____.
iii. char century [100][365][24][60[60] is an example of_____ array.
iv. An array of characters is called _____.
v. float x[12] defines one dimensional array of type _____.

Q.2 Select the correct choice for the following Multiple Choice Questions.

i. A collection of data items of the same data types placed in contiguous memory locations is called_____,
   a. Record
   b. Array
   c. Program
   d. File
ii. Observe the following statements and decide what they do.
   string mystring;
   getline(cin, mystring);
   a. reads a line of string from cin into mystring
   b. reads a line of string from mystring into cin
   c. cin can't be used this way
   d. none of the above
iii. Which of the header file must be included to use string stream?
   a. <iostream>
   b. <string>
   c. <sstring>
   d. <sstream>
iv. The location number of an item in an array is called _____.
   a. Data
   b. Value
   c. Index
   d. Constant