Arrays and Structures

Learning Objectives

After learning this chapter, the students will be able to

- Know the structured data type using arrays.
- Know the types of arrays.
- Writing programs to manuplates different types of arrays.

12.1 Introduction

The variables are used to store data. These variables are the one of the basic building blocks in C++. A single variable is used to store a single value that can be used anywhere in the memory. In some situations, we need to store multiple values of the same type. In that case, it needs multiple variables of the same data type. All the values are stored randomly anywhere in the memory.

For example, to store the roll numbers of the 100 students, it needs 100 variables named as roll1, roll2, roll3,……. roll100 . It becomes very difficult to declare 100 variables and store all the roll numbers. In C++, the concept of Array helps to store multiple values in a single variable. Literally, the meaning of **Array is "More than one". In other words, array is an easy way of storing multiple values of the same type referenced by a common name". An array is also a derived data type in C++.**

"An array is a collection of variables of the same type that are referenced by a common name". In an array, the values are stored in a fixed number of elements of the same type sequentially in memory. Therefore, an integer array holds a sequence of integers; a character array holds a sequence of characters and so on. The size of the array is referred to as its dimension.

12.2 Types of Arrays:

There are different types of arrays used in C++. They are:

- One-dimensional arrays
- Two-dimensional arrays
- Multi-dimensional arrays

12.2.1 One-dimensional array

This is the simplest form of an array. A one dimensional array represents values that are stored in a single row or in a single column.

Declaration

Syntax:

<data type><array_name> [<array_size>];

data_type declares the basic type of the array, which is the type of each element in the array.

array_name specifies the name with which the array will be referenced.

array_size defines how many elements the array will hold. Size should be specified with square brackets [ ].

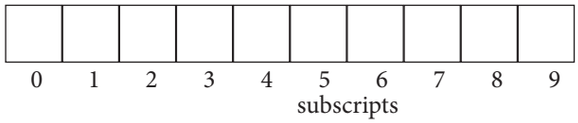
Example:

```
int num[10];
```

In the above declaration, an array named "num" is declared with 10 elements (memory space to store 10 different values) as integer type.

For the above declaration, the compiler allocates 10 memory locations (boxes) referenced by a common name "num" as given below

int num[10];



Each element (Memory box) has a unique index number starting from 0 which is known as "subscript". The subscript always starts with 0 and it should be an unsigned integer value. Each element of an array is referred by its name with subscript index within the square bracket. For example, num[3] refers to the 4th element in the array.

Some more array declarations with various data types:

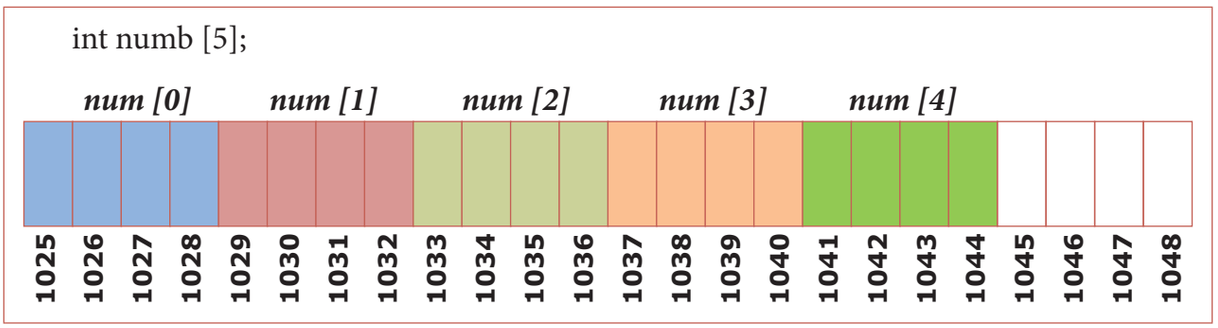
char emp_name[25]; // character array named emp_name with size 25

float salary[20]; // floating-point array named salary with size 20

int a[5], b[10], c[15]; // multiple arrays are declared of type int

Memory representation of an one dimensional array

The amount of storage required to hold an array is directly related with type and size. The following figure shows the memory allocation of an array with five elements.



The above figure clearly shows that, the array num is an integer array with 5 elements. As per the Dev-C++ compiler, 4 bytes are allocated for every int type variable. Here, there are totally 5 elements in the array, where for each element, 4 bytes will be allocated. Totally, 20 bytes will be allocated for this array.

Datatype	Turbo C++	Dev C++
char	1	1
int	2	4
float	4	4
long	4	4
double	8	8
long double	10	10

The memory space allocated for an array can be calculated using the following formula:

Number of bytes allocated for type of array × Number of elements

Initialization

An array can be initialized at the time of its declaration. Unless an array is initialized, all the array elements contain garbage values.

Syntax:

<datatype> <array_name> [size] = {value 1, value 2, ……, value n};

Example

int age[5]={19,21,16,1,50};

In the above example, the array name is 'age' whose size is 5. In this case, the first element 19 is stored in age[0], the second element 21 is stored in age[1] and so on as shown in figure 12.1

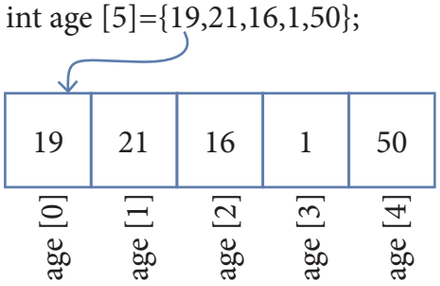


Figure 12.1

While declaring and initializing values in an array, the values should be given within the curly braces ie. { ….. }

The size of an array may be optional when the array is initialized during declaration.

Example:

int age[]={ 19,21,16,1,50};

In the above initialization, the size of the array is not specified directly in the declaration with initialization. So, the size is determined by compiler which depends on the total number of values. In this case, the size of the array is five.

More examples of array initialization:

float x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};

char vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};

Accepting values to an array during run time :

Multiple assignment statements are required to insert values to the cells of the array during runtime. The for loop is ideally suited for iterating through the array elements.

// Input values while execution

```
#include <iostream>
using namespace std;
int main()
{
        int num[5];
        for(int i=0; i<5; i++)
        {
                cout<< "\n Enter value " << i+1 << "= ";
                cin>>num[i];
        }
}
```

In the above program, a for loop has been constructed to execute the statements within the loop for 5 times. During each iteration of the loop, *cout* statement prompts you to "Enter value ……." and *cin* gets the value and stores it in num[i];

The following table shows the execution of the above code block.

Iteration	i <5	cout << "\n Enter value " << i+1 << "= ";	cin>>num [i];	Received value stored in memory		i++ (i=i+1)
1	5 > 0 (T)	Enter value 1 =	num[0] = 5	num[0]	5	1
2	5 > 1 (T)	Enter value 2 =	num[1] = 10	num[1]	10	2
3	5 > 2 (T)	Enter value 3 =	num[2] = 15	num[2]	15	3
4	5 > 3 (T)	Enter value 4 =	num[3] = 20	num[3]	20	4
5	5 > 4 (T)	Enter value 4 =	num[25 = [4	num[4]	25	5
6	5 > 5 (F)	Exit from Loop				

Note

In for loop, the index ***i*** is declared with an initial value 0 (zero). Since in most of the cases, the initial value of the loop index will be used as the array subscript representation.

Accessing array elements

Array elements can be used anywhere in a program as we do in case of a normal variable. The elements of an array are accessed with the array name followed by the subscript index within the square bracket.

Example:

```
cout<<num[3];
```

In the above statement, num[3] refers to the 4th element of the array and *cout* statement displays the value of num[3].

Note

The subscript in bracket can be a variable, a constant or an expression that evaluates to an integer.

// Accessing array elements

```
#include <iostream>
using namespace std;
int main()
{
        int num[5] = {10, 20, 30, 40, 50};
        int t=2;
        cout<<num[2] <<endl; // S1
        cout<<num[3+1] <<endl; // S2
        cout<<num[t=t+1]; // S3
}
```

output:

```
30
50
40
```

In the above program, statement **S1** displays the value of the 3rd element (subscript index 2). **S2** will display the value of the 5th element (ie. Subscript value is 3+1 = 4). In the same way statement **S3** will display the value of the 4th element.

C++ program to inputs 10 values and count the number of odd and even numbers

```
#include <iostream>
using namespace std;
int main()
{
        int num[10], even=0, odd=0;
        for (int i=0; i<10; i++)
         {
                cout<< "\n Enter Number " << i+1 <<"= ";
                cin>>num[i];
                if (num[i] % 2 == 0)
                 ++even;
                else
                 ++odd;
         }
         cout << "\n There are "<< even <<" Even Numbers";
         cout << "\n There are "<< odd <<" Odd Numbers";
}
```

Output:

```
Enter Number 1= 78
Enter Number 2= 51
Enter Number 3= 32
Enter Number 4= 66
Enter Number 5= 41
Enter Number 6= 68
Enter Number 7= 27
Enter Number 8= 65
Enter Number 9= 28
Enter Number 10= 94
There are 6 Even Numbers
There are 4 Odd Numbers
```

(HOT : Rewrite the above program using the conditional operator instead of if)

Searching in a one dimensional array:

Searching is a process of finding a particular value present in a given set of numbers. The linear search or sequential search compares each element of the list with the value that has to be searched until all the elements in the array have been traversed and compared.

Program for Linear Search

```
#include <iostream>
using namespace std;
int main()
{
        int num[10], val, id=-1;
        for (int i=0; i<10; i++)
        {
        cout<< "\n Enter value " << i+1 <<"= ";
        cin>>num[i];
        }
        cout<< "\n Enter a value to be searched: ";
        cin>>val;
    for (int i=0; i<10; i++)
    {
          if (arr[i] == val)
          {    id= i;
               break;
          }
    }
        if(id==-1)
        cout<< "\n Given value is not found in the array..";
        else
        cout<< "\n The value is found at the position" << id+1;
        return 0;
}
```

The above program reads an array and prompts for the values to be searched. It compares each element of the list with the value that has to be searched until all the elements in the array have been traversed and compared.

Strings

A string is defined as a sequence of characters where each character may be a letter, number or a symbol. Each element occupies one byte of memory. Every string is terminated by a null ('\0', ASCII code 0) character which must be appended at the end of the string. In C++, there is no basic data type to represent a string(However, we can use string object to holding strings in C++). Instead, it implements a string as an one-dimensional character array. When declaring a character array, it also has to hold a null character at the end, and so, the size of the character array should be one character longer than the length of the string.

Character Array (String) creation

To create any kind of array, the size (length) of the array must be known in advance, so that the memory locations can be allocated according to the size of the array. Once an array is created, its length is fixed and cannot be changed during run time. This is shown in figure12.2

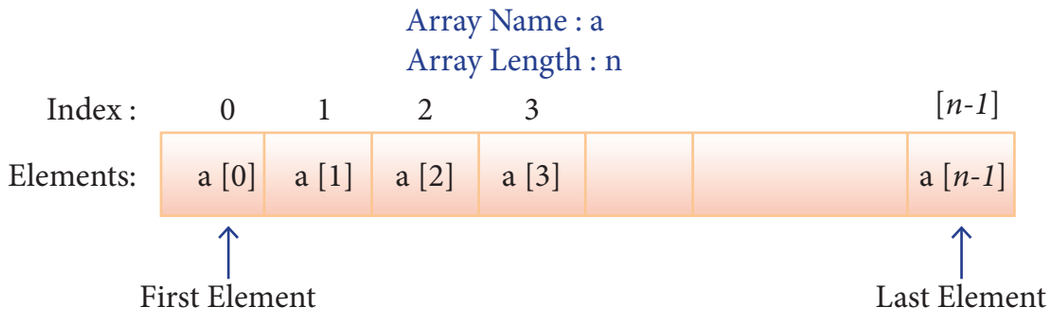


Figure 12.2

Syntax

Array declaration is:

char array_name[size];

In the above declaration, the size of the array must be an unsigned integer value.

For example,

char country[6];

Here, the array reserves 6 bytes of memory for storing a sequence of characters. The length of the string cannot be more than 5 characters and one location is reserved for the null character at the end.

//Program to demonstrate a character array.

```
#include <iostream>
using namespace std;
int main()
 {
        char country[6];
        cout<< "Enter the name of the country: ";
        cin>>country;
        cout<<" The name of the country is "<<country;
}
```

OUTPUT

```
Enter country the name: INDIA
The country name is INDIA
```

Initialization

The character array can be initialized at the time of its declaration. The syntax is shown below:

char array_name[size]={ list of characters separated by comma or a string } ;

For example,

char country[6]="INDIA";

In the above example, the text "INDIA" has 5 letters which is assigned as initial value to array country. The text is enclosed within double quotes. The memory representation is shown in Figure 12.3

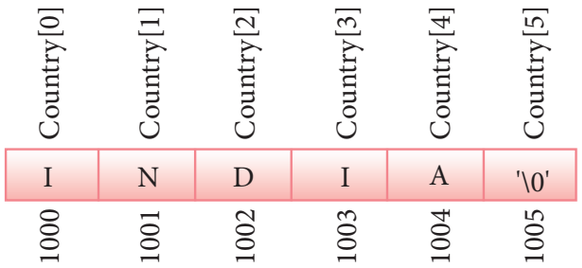


Figure 12.3

In the above memory representation, each character occupies one byte in memory. At the end of the string, a null character is automatically added by the compiler. **C++ also provides other ways of initializing the character array:**

char country[6]={'I', 'N', 'D', 'I', 'A', '\0'};

char country[]="INDIA";

char country[]={'I', 'N', 'D', 'I', 'A', '\0'};

If the size of the array is not explicitly mentioned, the compiler automatically calculate the size of the array based on the number of elements in the list and allocates space accordingly.

In the initialization of the string, if all the characters are not initialized, then the rest of the characters will be filled with NULL.

Example:

char str[5]={'5','+','A'};

str[0]; ---> 5

str[1]; ---> +

str[2]; ---> A

str[3]; ---> NULL

str[4]; ---> NULL

Note

During initialization, the array of elements cannot be initialized more than its size.

For example

char str[2]={'5','+','A','B'}; // Invalid

In the above example, the compiler displays "initialize-string for array of chars is too long" error message.

Write a Program to check palindrome or not

```
#include<iostream>
#include<string.h>
using namespace std;
int main( )
{
        int i, j, len, flag =1;
        char a [20];
        cout<<"Enter a string:";
        cin>>a;
        for(len=strlen(a);a[len]!='\0';++len)
        for(i=0,j=len-1;i<len/2;++i,--j)
        {
        if(a[j]!=a[i])
                        flag=0;
                        break;
        }
```

```
        if(flag==1)
                cout<<"\n The String is palindrome";
        else
                cout<<"\n The String is not palindrome";
        return 0;
}
```

Output:

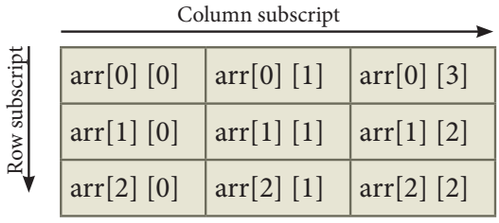
```
        Enter a string : madam
        The String is palindrome
```

12.3 Two-dimensional array

Two-dimensional (2D) arrays are collection of similar elements where the elements are stored in certain number of rows and columns. An example m × n matrix where m denotes the number of rows and n denotes the number of columns is shown in Figure12.4

int arr[3][3];

2D array conceptual memory representation



The array arr can be coneptually viewed in matrix form with 3 rows and 3 coloumns. The point to be noted here is since the subscript starts with 0 arr [0][0] represents the first element.

Figure 12.4

12.3.1 Declaration of 2-D array

The declaration of a 2-D array is

data-type array_name[row-size][col-size];

In the above declaration, data-type refers to any valid C++ data-type, array_name refers to the name of the 2-D array, row-size refers to the number of rows and col-size refers to the number of columns in the 2-D array.

For example

```
int A[3][4];
```

In the above example, A is a 2-D array, 3 denotes the number of rows and 4 denotes the number of columns. This array can hold a maximum of 12 elements.

Note

Array size must be an unsigned integer value which is greater than 0. In arrays, column size is compulsory but row size is optional.

Other examples of 2-D array are:

int A[3][3];

float x[2][3];

char name[5][20];

12.3.2 Initialization of Two-Dimensional array

The array can be initialized in more than one way at the time of 2-D array declaration.

For example

```
int matrix[4][3]={
{10,20,30},// Initializes row 0
{40,50,60},// Initializes row 1
{70,80,90},// Initializes row 2
{100,110,120}// Initializes row 3
};
int matrix[4][3]={10,20,30,40,50,60,70,80,90,100,110,120};
```

Array's row size is optional but column size is compulsory.

For example

```
int matrix[][3]={
{10,20,30},// row 0
{40,50,60},// row 1
{70,80,90},// row 2
{100,110,120}// row 3
};
```

12.3.3 Accessing the two-dimensional array

Two-dimensional array uses two index values to access a particular element in it, where the first index specifies the row value and second index specifies the column value.

matrix[0][0]=10;// Assign 10 to the first element of the first row

matrix[0][1]=20;// Assign 20 to the second element of the first row

matrix[1][2]=60;// Assign 60 to the third element of the second row

matrix[3][0]=100;// Assign 100 to the first element of the fourth row

Write a program to perform addition of two matrices

```
#include<iostream>
#include<conio>
using namespace std;
int main()
{
        int row, col, m1[10][10], m2[10][10], sum[10][10];
        cout<<"Enter the number of rows : ";
        cin>>row;
        cout<<"Enter the number of columns : ";
        cin>>col;
        cout<< "Enter the elements of first matrix: "<<endl;
        for (int i = 0;i<row;i++ )
        for (int j = 0;j <col;j++ )
        cin>>m1[i][j];
        cout<< "Enter the elements of second matrix: "<<endl;
        for (int i = 0;i<row;i++ )
        for (int j = 0;j<col;j++ )
        cin>>m2[i][j];
        cout<<"Output: "<<endl;
        for (int i = 0;i<row;i++ )
        for (int j = 0;j<col;j++ )
        {
        sum[i][j]=m1[i][j]+m2[i][j];
        cout<<sum[i][j]<<" ";
        }
        cout<<endl<<endl;
        }
getch();
return 0;
}
```

```
Enter the number of rows :   2
Enter the number of column : 2
Enter the elements of first matrix:
1
1
1
1
Enter the elements of second matrix:
1
1
1
1
Output:
2   2
2   2
```

12.3.4 Memory representation of 2-D array

Normally, the two-dimensional array can be viewed as a matrix. The conceptual view of a 2-D array is shown below:

int A[4][3];

A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

In the above example, the 2-D array name A has 4 rows and 3 columns.

Like one-dimensional, the 2-D array elements are stored in continuous memory.

There are two types of 2-D array memory representations. They are:

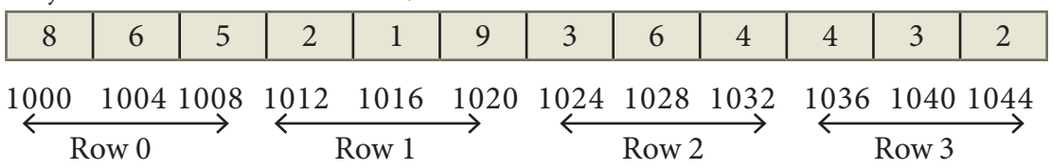
- Row-Major order
- Column-Major order

For example

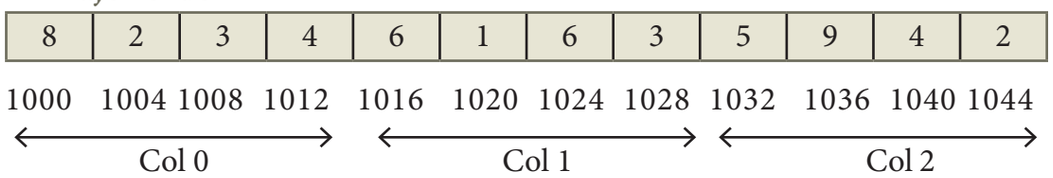
int A[4][3]={ { 8,6,5}, { 2,1,9}, {3,6,4}, {4,3,2} }

Row Major order

In row-major order, all the elements are stored row by row in continuous memory locations, that is, all the elements in first row, then in the second row and so on. The memory representation of row major order is as shown below;



Column Major order



12.4 Array of strings

An array of strings is a two-dimensional character array. The size of the first index (rows) denotes the number of strings and the size of the second index (columns) denotes the maximum length of each string. Usually, array of strings are declared in such a way to accommodate the null character at the end of each string. For example, the 2-D array has the declaration:

char Name[6][10];

In the above declaration, the 2-D array has two indices which refer to the row size and column size, that is 6 refers to the number of rows and 10 refers to the number of columns.

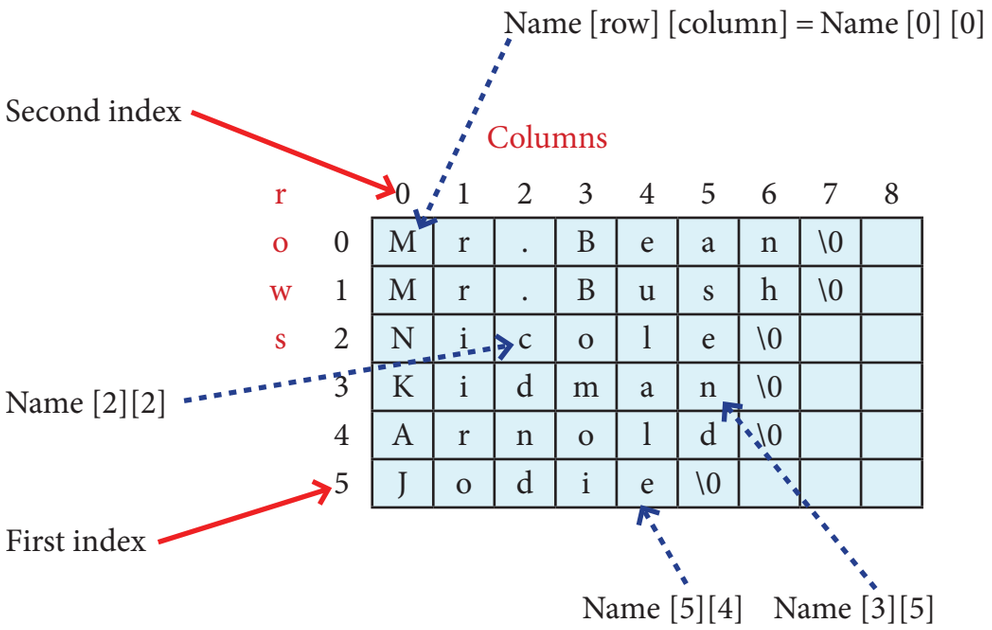
12.4.1 Initialization

For example

char Name[6][10] = {"Mr. Bean", "Mr.Bush", "Nicole", "Kidman", "Arnold", "Jodie"};

In the above example, the 2-D array is initialized with 6 strings, where each string is a maximum of 9 characters long, since the last character is null.

The memory arrangement of a 2-D array is shown below and all the strings are stored in continuous locations.



C++ program to demonstrate array of strings using 2-D character array

```
#include<iostream>
using namespace std;
int main( )
{
        // initialize 2d array
        char colour [4][10]={"Blue","Red","Orange",
                             "yellow"};

        // printing strings stored in 2d array
        for (int i=0;  i <4; i++)
        cout  <<  colour [i] << "\n";
}
```

Output:

```
Blue
Red
Orange
Yellow
```

Case Study:

(1) Write a program to accept the marks of 10 students and find the average, maximum and minimum marks.

Structures

12.5 Structures Introduction

Structure is a user-defined which has the combination of data items with different data types. This allows to group variables of mixed data types together into a single unit.

12.5.1 Purpose of Structures

In any situation when more than one variable is required to represent objects of uniform data-types, array can be used. If the elements are of different data types,then array cannot support. If more than one variable is used, they can be stored in memory but not in adjacent locations. It increases the time consumption while searching. The structure provides a facility to store different data types as a part of the same logical element in one memory chunk adjacent to each other.

12.5.2 Declaring and defining structures

Structure is declared using the keyword 'struct'. The syntax of creating a structure is given below.

```
struct structure_name {
        type member_name1;
        type member_name2;
} reference_name;
```

Objects declared along with structure definition are called global objects

An optional field reference_name can be used to declare objects of the structure type directly.

Example:

```
struct Student
{
        long rollno;
        int age;
        float weight;
} ;
```

In the above declaration of the struct, three variables rollno,age and weight are used. These variables(data element)within the structure are called members (or fields). In order to use the Student structure, a variable of type Student is declared and the memory allocation is shown in figure 12.5

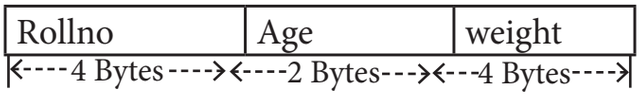


Fig 12.5 Memory Allocation

struct Student balu; // create a Student structure for Balu

This defines a variable of type Student named as Balu. Similar to normal variables, struct variable allocates memory for that variable itself. It is possible to define multiple variables of the same struct type:

struct Student frank; // create a structure for Student Frank.

For example, the structure objects balu and frank can also be declared as the structure data type as:

```
struct Student
{
long rollno;
int age;
float weight;
} balu, frank;
```

12.5.3 Referencing Structure Elements

Once the two objects of student structure type are declared (balu and frank),their members can be accessed directly. The syntax for that is using a dot (.) between the object name and the member name. For example, the elements of the structure Student can be accessed as follows:

```
balu.rollno
balu.age
balu.weight
frank.rollno
frank.age
frank.weight
```

(Anonymous Structure Vs Named Structure)
A structure without a name/tag is called anonymous structure.

```
struct
{
long rollno;
        int age;
        float weight;
} student;
```

The student can be referred as reference name to the above structure and the elements can be accessed like student.rollno, student.age and student.weight .

12.5.4 Initializing structure elements

Values can be assigned to structure elements similar to assigning values to variables.

Example

```
balu.rollno= "702016";
balu.age= 18;
balu.weight= 48.5;
```

Also, values can be assigned directly as similar to assigning values to Arrays.

```
balu={702016, 18, 48.5};
```

12.5.5 Structure Assignment

Structures can be assigned directly instead of assigning the values of elements individually.

Example

If Mahesh and Praveen are same age and same height and weight then the values of Mahesh can be copied to Praveen

```
struct Student
{
        int  age;
        float height, weight;
}mahesh;
```

Structure assignment is possible only if both structure variables/ objects are same type.

The age of Mahesh is 17 and the height and weights are 164.5 and 52.5 respectively.The following statement will perform the assignment.

```
mahesh = {17, 164.5, 52.5};
praveen =mahesh;
```

will assign the same age, height and weight to Praveen.

Examples:

The following C++ program reads student information through keyboard and displays the same

```
#include <iostream>
using namespace std;
struct Student
{
        int  age;
        float height, weight;
} mahesh;
void main( )
{
        cout<< " Enter the age:"<<endl;
        cin>>mahesh.age;
        cout<< "Enter the height:"<<endl;
        cin>>mahesh.height;
        cout<< "Enter the weight:"<<endl;
        cin>>mahesh.weight;
        cout<< "The values entered  for Age, height and weight are"<<endl;
        cout<<mahesh.age<<     "\t"<<mahesh.height<<     "\t"<<Mahesh.
weight;
}
```

Output:

```
Enter the age:
18
Enter the height:
160.5
Enter the weight:
46.5
The values entered  for Age, height and weight are
18       160.5            46.5
```

Points to Remember:

- Structure is a user-defined which has the combination of data items with different data types
- Structure is declared using the keyword 'struct'
- Structure elements are referenced using its object name followed by dot(.) operator and then the member name
- A structure without a name/tag is called anonymous structure.
- The structure elements can be initialized either by using separate assignment statements or at the time of declaration by surrounding its values with braces.
- A structure object can also be assigned to another structure object only if both the objects are of same structure type.
- The structure declared within another structure is called a nested structure
- A structure can contain array as its member element.
- Array of structure variable can also be created.