Arrays

- It is a collection of data items grouped into a single unit.
- Data items must be of same data type
- Data type can be int, float, double or user defined data type like class and structure
- Data items in structure are accessed by name , in arrays they are accessed by index number(to be discussed later)

Arrays

- It contains ordered list of values
- Items are indexed from 0-n-1 elements
- Concept of an array helps us to store group of values under a single name and refer directly to a particular value, which makes program simple and efficient

Declaration

- Syntax for declaring an array
- Data type array_name[maximum size]
- it means int a[10];
- Array is of integer data type and a is the name of an array and maximum elements it can hold are 10.

Storage Space allocation

Consider an example int marks[10];

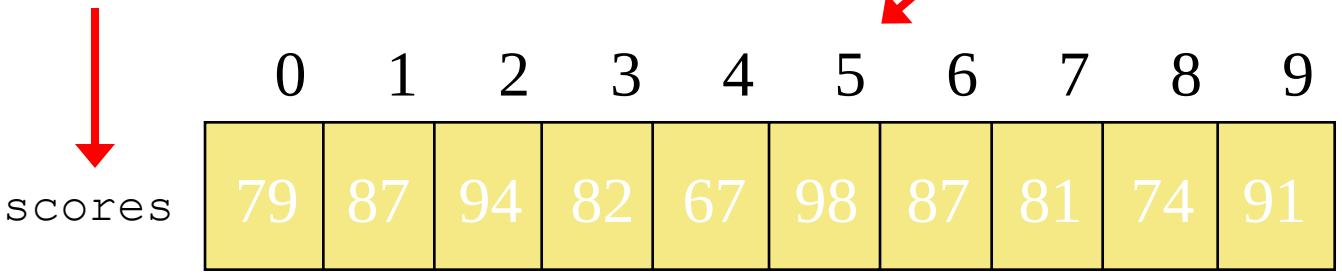
- Marks is an array of integer data type that can store the marks of 10 students..
- The storage space allocated = maximum number of elements multiplied with the size of the data type. All array elements are stored in contiguous memory location i.e one after another

Arrays

An *array* is an ordered list of values

The entire array has a single name

Each value has a numeric *index*



An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Accessing Array Elements

- We can access the array elements by using the name of an array followed by index enclosed between pair of square brackets.
- Index or subscript of an array indicates the position of element of an array
- Index of 1st element of an array is 0 and 2nd element is 1
- To assign value to an element of one dimensional array
- Array_name[index]=value;

Simple Program

```
#include <iostream.h>
#include<conio.h>
int main()
{
int age[4]; //array ‘age’ o
for(int j=0; j<4; j++) //get 4 ages
{
cout << “Enter an age: “;
cin >> age[j]; //access array element
}
for(j=0; j<4; j++) //display 4 ages
cout << “You entered “ << age[j] << endl;
return 0;
}
```

- The first for loop gets the ages from the user and places them in the array, while the second
- reads them from the array and displays them.

Initialization

Syntax for array initialization is

Data_type array_name[size]={value1, value2……}

1. int a[5]={1,2,3,4,5};

Here 1 is assigned to a[0],2 is assigned to a[1]…

2.char a[10]={‘a’,’b’,’c’,’d’,’-’};

Once array has been created , its size has been fixed , it can’t be changed

For example- int a[10];

we can’t insert 11^{th} element to a[11] location

Initialization

- Size of an array can be omitted during initialization. The compiler automatically determines the size of an array by number of elements inside it

  int []={12,2,3}; is a valid statement

- If size is specified , number of elements present in an array should not be greater than size. For example

  int a[2]={1,2,3};

Initialization

- If size is greater than number of elements then rest of the elements are initialized to 0.
- For example

int a[5]={1,2,3,4};

a[4]=0; by default

Passing Arrays to a function

Following Points must be required when we are passing arrays to functions

- Array name must be specified without any square brackets as an actual argument within function call.
- Formal Arguments must be declared as an array whose size need not be specified in formal argument declaration, but empty pair of square brackets must be specified with array name

Passing Arrays to function

- While passing arrays to function, called function does not know about array size and neither does compiler know about that, we need to pass array size as separate argument so that function knows number of elements to be processed
- Example

int x[10];

Function call will be- function_name(x,10);

Its definition will be

```
Void function_name( int x[],int n)
{
}
```

Function definition contains to formal argument declaration, first one is array declaration without size and integer declaration for number of elements in the array

Syntax

```
main()
{
int x[10];
 void sum(int [], int );
sum  (x,10);
void sum(int a[], int n)
{
Logic
}
```

Two Dimensional Array

- It is represented as int marks[2][3];

  It is used to store the marks of of 2 students in 3 subjects.
- It is used to store the values in the form of tables in such a way that information is arranged in the form of rows and columns.
- Row starts from 0 and ends with 1 and column starts with 0 and end with 2

2 Dimensional array

- The total number of elements can be determined by product of rows and columns
- Storage space allocation= No. of elements multiplied with size of the elements
- Element of 1^{st} row and 1^{st} column can be accessed by a[0][0],1^{st} row and 2^{nd} column=a[0][1].

Accessing elements in 2 dimensional Array

- Elements can be accessed by use of nested for loops.
- Elements of 2 dimensional array can be accessed for int marks[2][3] by:

```
for(int i=0;i<2;i++)
for(int j=0;j<3;j++)
cin>>marks[i][j];
```

Program to Initialize elements in 2 Dimensional Array

```
#include<iostream.h>
#include<conio.h>
main()
{
    int a[10][10],m,n,i,j;
    cout<<"Enter the Number of Rows and Columns:";
    cin>>m>>n;
    cout<<"enter the elements of an array:";
    for( i=0;i<m;i++)
    for( j=0;j<n;j++)
    cin>>a[i][j];
    cout<<"elements are:\n";
    for(i=0;i<m;i++)
    {
    for(j=0;j<n;j++)
    cout<<a[i][j]<<"\t";
    cout<<"\n";
    }
    getch();
    }
```

Initialization of 2 Dimensional array

- 2 Dimensional Arrays are initialized in the same way as we initialize elements in 1 D array.
- int marks[2][3]={3,5,6};
- Here int marks[0][0]=3, int marks[0][1]=5, int marks[0][2]=6, and rest other elements are initialized to 0;

Other Types of Initialization of 2 Dimensional Array

- int a[2][2]={{2},{3}}; here a[0][0]=2,a[1][0]=3;
- int a[2][2]={2,3}; here a[0][0]=2,a[0][1]=3;
- Int a [2][2]={2,3,4,5};// correct
- int a [2][]={2,3,4,5};// correct
- int a [2][]={2,3,4,5};// incorrect
- int a [][]={2,3,4,5};// incorrect

  It is necessary to mention the size specification of second subscript of 2-D array where 1st size specification is optional while initialization of an array

Multi Dimensional Arrays

- An array with more than 1 dimension is known as multidimensional array. An array may consist of any number of dimensions . It depends upon compiler how much it allows.
- Syntax:Data_Type array_name[size1][size2]……

…….[size m];

- Example- int a[3][4][5];
- Number of elements=3*4*5=60;
- To access all the elements of a multidimensional array, the number of for loops used depends upon dimensions of an array.

Multi Dimensional Arrays

```
#include<iostream.h>
#include<conio.h>
main()
{
    int a[2][2][2],m,n,s,i,j,k;
    cout<<"Enter the Number of Rows and Columns, sides:";
    cin>>m>>n>>s;
    cout<<"enter the elements of an array:";
    for( i=0;i<m;i++)
    for( j=0;j<n;j++)
    for(k=0;k<s;k++)
    cin>>a[i][j][k];
    cout<<"elements are:\n";
    for(i=0;i<m;i++)
    {
    for(j=0;j<n;j++)
    for(k=0;k<s;k++)
    cout<<a[i][j][k]<<"\t";
    cout<<"\n";
    }
    getch();
    }
```

Strings

- Collection of characters
- Array of Characters
- Store words and sentences.
- Collection of characters terminated by null character(‘\0’)
- Each element is stored in memory location , string is terminated by null character
- Enclosed in double quotes, That can include alphabets , numbers, operators, blank spaces.

Strings(Declaration and Initialization)

String is declared as

char name[20];

it means it can store up to 19 characters where as 20^{th} will be null character.

Initialization

char name[7]=“chirag”;

Or

char name[7]={‘c’,’h’,’i’,’r’,’a’,’g’,’\0’};

But storing this char name[6]= “chirag”; will provide some unpredictable results

Strings

- char name[10]=“chirag ”; is also possible but remaining charcters will be filled with null characters so it will be wastage of memory
- They can also be initialized by

  char name[]=“chirag”;
- Compiler will automatically assign required memory to the string, it sets 7 for this string, This method is widely for string containing large number of charcters

Inputting Multi-Word String

- Whenever we perform inputting operation using extraction operator, for eg: cin>>chirag Sharma, It will skip the characters after the white space
- So multiword is skipped by cin>>, to avoid this we use cin.get()
- It takes 2 arguments, string name and maximum size

Inputting Multi-line String

- Multi-line strings can also be input by the user with some modifications
- Syntax: cin.getline(name,max,delim)
- Name specifies name of string, max: maximum number of characters, delim is the character when to stop reading.

C++ Standard String Handling Functions

- strlen(Finding string Length)
- strcat(Appends one string at the end of another)
- strcmp(comparing 2 strings)
- strrev(reversing a string)
- strlwr(converting string to lowercase)
- strupr(converts string to upper case)

Array of Strings

- An array of strings can be represented by 2 Dimensional array where 1st specification tells the number of strings and second one tells number of characters in each string.
- char city[4][10];
- For example: city is declared as 2 dimensional array that can have 4 strings and maximum number of characters in each string can be 10

Accessing from array of strings

- To access a particular string from array of strings, specify only 1st subscript of the array
- cout<<city[1]; it display 2nd second city
- To access a character of a string from an array of strings, specify both the subscripts of the array.
- 1st subscript represents a string and 2nd represents character of the string, example :
- char city[2][3] represents 4th character in 3rd string in an array

Array of Strings

Arrays of Strings

If there are arrays of arrays, of course there can be arrays of strings. This is actually quite a useful construction. Here's an example, STRARAY, that puts the names of the days of the week in an array:

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

int main()
   {
   const int DAYS = 7;              //number of strings in array
   const int MAX = 10;              //maximum size of each string
                                    //array of strings
   char star[DAYS][MAX] = { "Sunday", "Monday", "Tuesday",
                            "Wednesday", "Thursday",
                            "Friday", "Saturday"  };
   for(int j=0; j<DAYS; j++)        //display every string
      cout << star[j] << endl;
   return 0;
   }
```

Array of Strings

The program prints out each string from the array:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

Since a string is an array, it must be true that `star`—an array of strings—is really a two-dimensional array. The first dimension of this array, `DAYS`, tells how many strings are in the array. The second dimension, `MAX`, specifies the maximum length of the strings (9 characters for `"Wednesday"` plus the terminating null makes 10).

Standard C++String Class

The Standard C++ `string` Class

Standard C++ includes a new class called `string`. This class improves on the traditional C-string in many ways. For one thing, you no longer need to worry about creating an array of the right size to hold string variables. The `string` class assumes all the responsibility for memory management. Also, the `string` class allows the use of overloaded operators, so you can concatenate string objects with the + operator:

```
s3 = s1 + s2
```

There are other benefits as well. This new class is more efficient and safer to use than C-strings were. In most situations it is the preferred approach. (However, as we noted earlier, there are still many situations in which C-strings must be used.) In this section we'll examine the `string` class and its various member functions and operators.

Defining and Assigning string Objects

```
#include <iostream.h>
#include <string.h>
 main()
{
string s1("Man"); //initialize
string s2 = "Beast"; //initialize
string s3;
s3 = s1; //assign
cout << "s3 = " << s3 << endl;
s3 = "Neither " + s1 + " nor "; //concatenate
s3 += s2; //concatenate
cout << “s3 = “ << s3 << endl;
s1.swap(s2); //swap s1 and s2
cout << s1 << " nor " << s2 << endl;
getch();
}
```

Input and Output with String Objects

```
#include <iostream>
#include <string> //for string class
#include<conio.h>
using namespace std;
 main()
{ //objects of string class
string full_name, nickname, address;
string greeting("Hello ");
cout << "Enter your full name: ";
getline(cin, full_name); //reads embedded blanks
cout << "Your full name is: " << full_name << endl;
cout << "Enter your nickname: ";
cin >> nickname; //input to string object
greeting += nickname; //append name to greeting
cout << greeting << endl; //output: “Hello, Jim”
cout << "Enter your address on separate lines\n";
cout << "Terminate with ‘$’\n";
getline(cin, address, '$'); //reads multiple lines
cout << "Your address is: " << address << endl;
getch();
}
```

Operations on String Objects

- Find
- Find_First_of
- Find_First_not_of
- Erase
- Replace
- Insert
- Append
- Compare
- Size
- Length