

# Lesson 7: Arrays and Strings

---

## Objectives

---

After completing this lesson you will understand about:

- The basics of defining, declaring, initializing and using arrays.
- Passing indexed variables and entire as arguments to functions.
- The arrays of classes and classes with arrays as member variables.
- The basics of multidimensional arrays, multidimensional array parameters and passing them into functions.
- The string basics, declaring and initializing cstring variables, predefined cstring functions and getline function.
- Defining cstring functions and array of strings.

---

## Structure Of The Lesson

---

- 7.1 Introduction to arrays
- 7.2 Arrays in functions
  - 7.2.1 Indexed variables as function arguments
  - 7.2.1 Passing entire arrays as function arguments
  - 7.3 Arrays and classes
- 7.4 Multidimensional arrays
  - 7.4.1 Accessing multidimensional arrays
  - 7.4.2 Initialization of Multidimensional arrays
  - 7.4.3 Storage of two dimensional arrays
  - 7.4.4 Passing multidimensional arrays into functions
- 7.5 Strings
  - 7.5.1 Index variables of cstrings
  - 7.5.2 Operations on cstrings
  - 7.5.3 cstring predefined functions
  - 7.5.4 Defining cstring functions
  - 7.5.5 Arrays of strings
- 7.6 C++ Standard string class
- 7.7 Summary
- 7.8 Technical Terms
- 7.9 Model questions
- 7.10 References




---

## 7.1 Introduction To Arrays

---

An array is a group of related data items of same data type that share a common name. It is used to process a collection of data, all of which is of the same type such as list of test scores, a list of temperatures, list of names etc.

**Syntax:**

Base type	name	Size of array
		
datatypename	arrayname	[size]

An array declaration of the above form will define the declared size number of index variables namely `arrayname[0]`, `arrayname[1]`,..... `arrayname[size -1]`. Each index variable is a variable of type `datatypename`. These index variables are also known as subscripted variables or elements. The number in the square bracket is called as index number or subscript. Indexes are numbered starting with zero. The number of index variables in an array is known as the size of the array. The data type of the array is called the base type of the array.

**e.g.:**

```
int score[100];
double temp[50];
```

`score` is an integer array consisting of 100 elements starting from `score[0]` to `score[99]`. As it is an integer array, integer type of data can be stored in it.

**Initializing The Arrays:** An array variable can be initialized like any other variable. When initializing an array, the values of various index variables are enclosed in braces and separated with commas. All the values in the list should be of the base type of the array. This list is called an initialized list.

**e.g.:**

```
int score[3]={5,10,50};
```

In the above example, it is array called `score` with three elements where `score[0]=5`, `score[1]=10`, `score[2]=50`. If a fewer values than the size of the array are declared, those values will be used to initialize the first few index variables and remaining will be initialized to zero of array base type.

**e.g.:**           int score[5]={5,10,50};  
                  Here score[3] and score[4] are initialized to zeros.

If an array is initialized when it is declared the size of the array can be omitted. The array will be automatically declared to have the minimum size needed for the initialized variables. C++ will create the array with sufficient size to accommodate the entire initialized list.

**e.g.:**           int score[] = { 5,10,15};

Here, the size of the array is not given. But, depending upon the number of arguments present in the array during initialization, the array is declared to be of size 3 automatically.

**Write a program to read five scores and show how much each score differs from the highest score.**

```
#include<iostream.h>
#include<conio.h>
int a[20];//declaration of an array
int main()
{
int i,size, max=-20;//Take max as a small value
//clrscr();
cout<<"enter size of array:";
cin>>size;
cout<<" enter "<<size<<" scores";
for(i=0;i<size;i++)
cin>>a[i];
for(i=0;i<size;i++)
if (a[i]>max)
max=a[i];
cout<<"the difference from max marks:";
for(i=0;i<size;i++)
cout<<"a["<<i<<"]="<<a[i]<<"\t"
<<"Difference:"<<max-a[i]<<"\n";
getch();
return 0;
}
```

**Output:**

```
enter size of array:5
enter 5 scores
10
20
```

30  
40  
50  
the difference from max marks:  
a[0]=10 Difference:40  
a[1]=20 Difference:30  
a[2]=30 Difference:20  
a[3]=40 Difference:10  
a[4]=50 Difference:0

---

## 7.2 Arrays In Functions

---

In this section we will study how array elements are sent as arguments to the functions as well as how an entire array is passed as an argument to the function.

---

### 7.2.1 Indexed Variables As Function Arguments

---

An indexed variable can be an argument to a function in exactly the same way that any variable can be argument.

**eg:**           int i,n,a[10];

The variables of an array can be passed into a function as an individual data item.

Here is a sample program:

**// Program to pass array element as an argument to the function**

```
#include<iostream.h>
int diff(int,int);
int main()
{
int a[5],i;
int max=-20;
for(i=0; i<5;i++)
{
cout<<"enter score of each student in one subject:";
cin>>a[i];
}
for(i=0; i<5;i++)
```

```

if (a[i]>max)
max=a[i];
cout<<"The maximum score is:"<<max<<"\n";
cout<<" The scores and their difference from the
maximum score:\n";
int d;
for(i=0;i<5;i++)
{
d=diff(a[i],max);
cout<<a[i]<<"\t"<<d<<"\n";
}
return 0;
}
int diff (int x, int m)
{
return(m-x);
}

```

**output:**

```

enter score of each student in one subject:10
enter score of each student in one subject:20
enter score of each student in one subject:30
enter score of each student in one subject:60
enter score of each student in one subject:50
The maximum score is:60
The scores and their difference from the maximum
score:
    10   50
    20   40
    30   30
    60   10
    50   10

```

**Write a program to add five to each employees allowed no.of vacations.**

```

#include<iostream.h>
int adjustdays(int old_days);
void main()
{
int i,vacation[5],emp;
cout<<"enter no of employees:\n";
cin>>emp;
cout<<"Enter leave details for "<<emp<<" employees";

```

```

for(i=0;i<emp;i++)
{
cin>>vacation[i];
vacation[i]=adjustdays(vacation[i]);
}
cout<<"vacations after updation:\n";
for(i=0;i<emp;i++)
{
cout<<"employee:"<<i+1<<"vacation details "<<
vacation[i]<<"days \n";
}
}
int adjustdays(int olddays)
{
return(olddays+5);
}

```

**output:**

```

enter no of employees:
3
Enter leave details for 3 employees
2
4
1
vacations after updation:
employee:1 vacation details 7days
employee:2 vacation details 9days
employee:3 vacation details 6days

```

---

## 7.2.2 Passing Entire Arrays As Function Arguments

---

An argument to a function may be an entire array, but an argument for the entire array is neither a call by value nor a call by reference argument. It is a new kind of argument known as array argument. When an array argument is plugged in for an array parameter, all that is passed to the function is the address in memory of first index variable of the array argument (i.e., one indexed by 0).

The array argument does not tell the function the size of the array. Therefore when an array parameter is passed to a function, normally another formal parameter of type int that gives the size of the array should be present.

An array argument is like a call by reference argument. If the function body changes the array parameter, when the function is called, these changes are actually made to the array arguments. Thus, the function can change the values of an array argument.

### **Syntax of function prototype passing an array parameter:**

Returntype functionname(basetype arrayname[ ]...);

e.g.:                void sumarray (double & sum,double a[],int size);

**Write a program to read an array and find the sum of elements in an array.**

```
#include<iostream.h>
void read(int a[], int &size);
int sum(int a[], int&size);
void display (int ,int );
void main()
{
int a[20],num,s;
read(a,num);
s=sum(a,num);
display(s,num);
}
void read (int a[],int &size)
{
cout<<"enter size of an array:";
cin >> size;
cout<<"Enter the elements:";
for(int i=0;i<size;i++)
cin>>a[i];
return;
}
int sum (int a[],int &size)
{
int s=0;
for (int i=0;i<size;i++)
s+=a[i];
return s;
}
void display (int s, int size)
{
cout<<"the sum of "<<size<<" elements is"<<s;
}
}
```

**output:**

```
enter size of an array:5
2
3
4
5
6
the sum of 5elements is20enter size of an array:5
Enter the elements:1
2
3
4
5
the sum of 5 elements is15
```

**Write a program to search a number using linear search technique.**

```
#include<iostream.h>
enum bool{false,true};
int search(int const arr[],int,int);
void read (int a[],int size);
int main()
{
int num,arr[30],result,target;
cout<<"enter size:";
cin >> num;
cout<<"enter elements:";
read(arr,num);
cout<<"enter element to search";
cin >>target;
result=search(arr,num,target);
if(result== -1)
cout << target<<"not found";
else
cout << target <<" found at"<<(result+1);
return 0;
}
void read (int a[],int num)
{
for(int i=0;i<num;i++)
cin>>a[i];
}
int search(int const arr[],int num,int tar)
{
```



```

bool found=false; int i=0;
while((!found)&&(i<num))
{
    if (tar==arr[i])
found=true;
else
i++;
}
if(found==true)
return i;
else
return -1;
}

```

**output:**

```

enter size:4
enter elements:
2
6
4
1
enter element to search4
4 found at3

```

**Program to sort the given integers using selection sorting technique.**

```

#include<iostream.h>
void read (int a[],int &size);
void display (int a[],int &size);
void swap(int& v1,int& v2);
void main()
{
int arr[20],numb;
int minindex,min;
read(arr,numb);
cout<<" Array before sorting:\n";
display(arr,numb);
int i;
for(i=0;i<numb-1;i++)
{
min=arr[i];
minindex=i;

```

```

for (int j=i+1;j<numb;j++)
if(arr[j]<min)
{
min=arr[j];
minindex=j;
}
swap(arr[i],arr[minindex]);
}
cout<<"\nArray after sorting:\n";
display (arr,numb);
}
void read (int a[],int &size)
{
cout<<"enter size of an array:";
cin >> size;
for(int i=0;i<size;i++)
cin>>a[i];
return;
}
void swap(int& v1,int& v2)
{
int temp;
temp = v1;
v1 = v2;
v2 = temp;
}
void display (int a[],int &size)
{
for(int i=0;i<size;i++)
cout<<a[i]<<"\n";
return;
}

```

**output:**

```

enter size of an array:4
5
3
6
2
Array before sorting:
5
3
6
2
Array after sorting:

```

2  
3  
5  
6

---

## 7.3 Arrays And Classes

---

We can also declare the arrays of classes and classes with arrays as member variables.

**Arrays of classes:** The base type of an array can be a class. We can declare an array of variable of that class type then each array element will be an object of the class. The syntax for declaring an array of object is

**Syntax:**            **Classname arrayname[size];**  
e.g:                 student s1[10];

### Program to illustrate the arrays of objects

```
#include <iostream.h>
class employee
{
    char name[30];
    float age;
public:
    void getdata(void);
    void putdata(void);
};
void employee::getdata(void)
{
    cout<<"Enter name";
    cin>>name;
    cout<<"Enter age";
    cin>>age;
}
void employee::putdata(void)
{
    cout<<"Name: "<<name<<"\n";
    cout<<"Age: "<<age<<"\n";
}
const int size = 2;
```

```

int main( )
{

    employee manager[size];//Array of managers
    int i;

    for (i = 0; i < size; i++)
    {
        cout<<"\nDetails of manager"<<i+1<<"\n";
        manager[i].getdata();
    }
    cout<<"\n";
    for (i = 0; i < size; i++)
    {
        cout<<"\nManager"<<i+1<<"\n";
        manager[i].putdata();
    }
    return 0;
}

```

**Output:**

```

Details of Manager1
Enter name:harish
Enter age:40

```

```

Details of Manager1
Enter name:girish
Enter age:42

```

```

Manager1
Name :harish
Age:40

```

```

Manager2
Name :girish
Age:42

```

**Array as class members:** Arrays can be declared as members of a class.They can be of any data type. The arrays within the class can be private, public or protected. Then every object of that class will contain a copy of array declared in the class.

```

#include<iostream.h>
class student
{

```

```

char name[20];
int rno;
int marks[3];
public:
void read()
{
    cout<<"Enter name:";
    cin>>name;
    cout<<"Enter rno:";
    cin>>rno;
    cout<<"Enter 3 marks:";
    for(int i = 0;i<3;i++)
    {
        cin>>marks[i];
    }
}
void display()
{
    cout<<"Name: "<<name<<"\n";
    cout<<"RNo: "<<rno<<"\n";
    for(int i = 0;i<3;i++)
    cout<<marks[i]<<"\t";
    cout<<"\n";
}
};

void main()
{
    student stu;
    stu.read();
    stu.display();
}

```

**Output:**

```

Enter name:rama
Enter rno:1
Enter 3 marks:90
80
70
Name: rama
RNo: 1
90  80  70

```

---

## 7.4 Multidimensional Arrays

---

It is sometimes useful to have more than one index, and this is allowed in C++ with multidimensional arrays. A multidimensional array is an array of arrays. An array with more than one subscript or size specifier is defined as multidimensional array.

**Syntax:**        Base type dataname[size 1][size 2]...[size n];  
e.g.:             int mat[2][3];

In the above example, mat is a two dimensional matrix with 2 rows and 3 columns. The index variables of the array mat are:

mat[0][0] , mat[0][ 1], mat[0][2]  
mat[1][0] , mat[1][ 1], mat[1][2]

The number of elements in a multi dimensional array is equal to the product of all its subscripts. Thus the above matrix has 2X 3 elements. The indexes start from 0 to size-1.

---

### 7.4.1 Accessing Multidimensional Array Elements

---

Two indices or subscripts are used to access two-dimensional arrays, three for three-dimensional arrays and so on. The values of the elements of the two dimensional arrays are accessed by specifying the name of the variable, row and column index of the element.

---

### 7.4.2 Initialization Of Multidimensional Arrays

---

Consider the array marks[5][4]. It is an integer array. The first subscript indicates rollno of a student, second subscript indicates marks obtained by each student in 4 subjects. Thus this array has 5 rows, 4 columns and with subscripts 0 to 4 & 0 to 3. This array can be initialized as

```
int marks[5][4]={{20,40,60,70}, {3,50,40,60} ,{4,9,10,16},  
                {10,,60,20,30} , {50,80,60,90}};
```

---

### 7.4.3 Storage of two dimensional arrays

---

The two dimensional arrays are stored in two ways. They are row major order, column major order.

For e.g. : let us consider a 3x4 integer matrix, mat. The different indexed variables of the matrix is as follows:

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

Let us see how this matrix is stored in the computers memory. The computer's memory consists of list of numbered location called bytes. The variable is represented as a portion of this memory. Thus, a variable is described by two pieces of information: an address in memory (giving the location of the first byte for that variable) and the type of the variable. The location of various array-indexed variables are always placed next to one another in memory. When the array is declared, the computer reserves enough memory to hold the variables of the array, depending on the data type. The computer remembers the address of the first variable mat[0][0].

Let us consider the row major order. There are four columns the starting address of k th row will be

Base + k th row x no. of columns x size of the data type

Let the first element of the array is stored at the address location 200. For an element a[2][0] it is stored that

$$200+2*4*2$$
$$200+16=216$$

To get the full address of an array variable we use

base+(rowindex X total no.of columns) X size of the datatype) + columnindex\* size of data type.

Similarly in column major order the storage of a[i][j] is given as

Base + (columnindex X total no of rows) \* size of datatype)+ rowindex X size of datatype

---

## 7.4.4 Passing Multidimensional Arrays Into Functions

---

When a multidimensional array parameter is given in a function heading or prototype, the size of the first dimension is not given, but the remaining dimension sizes must be given in square brackets. Since the first dimension size is not given, an additional parameter of type `int`, which gives the size of the first dimension is needed. Here is an example of a function prototype with a two dimensional array parameter:

```
//Program to read and display the quiz scores of given students
void readarray(int p[][100], int sizedimension);

#include <iostream.h>
#include <iomanip.h>
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;
void compute_st_ave(const int grade[][NUMBER_QUIZZES], double
st_ave[]);
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double
quiz_ave[]);
void display(const int grade[][NUMBER_QUIZZES],
              const double st_ave[], const double quiz_ave[]);

int main( )
{
    int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
    double st_ave[NUMBER_STUDENTS];
    double quiz_ave[NUMBER_QUIZZES];

    grade[0][0] = 10; grade[0][1] = 10; grade[0][2] = 10;
    grade[1][0] = 2; grade[1][1] = 0; grade[1][2] = 1;
    grade[2][0] = 8; grade[2][1] = 6; grade[2][2] = 9;
    grade[3][0] = 8; grade[3][1] = 4; grade[3][2] = 10;

    compute_st_ave(grade, st_ave);
    compute_quiz_ave(grade, quiz_ave);
    display(grade, st_ave, quiz_ave);
    return 0;
}

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double
st_ave[])
```



```

{
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
        { //Process one st_num:
            double sum = 0;
            for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
                sum = sum + grade[st_num-1][quiz_num-1];
            //sum contains the sum of the quiz scores for student numberst_num.
            st_ave[st_num-1] = sum/NUMBER_QUIZZES;
            //Average for student st_num is the value of st_ave[st_num-1]
        }
}
}
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double
quiz_ave[])
{
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
        { //Process one quiz (for all students):
            double sum = 0;
            for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
                sum = sum + grade[st_num-1][quiz_num-1];
            //sum contains the sum of all student scores on quiz number quiz _
            num.
            quiz_ave[quiz_num-1] = sum/NUMBER_STUDENTS;
            //Average for quiz quiz_num is the value of quiz_ave[quiz_num-1]
        }
}
}

//Uses iostream and iomanip:
void display(const int grade[][NUMBER_QUIZZES],
             const double st_ave[], const double quiz_ave[])
{
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);

    cout << setw(10) << "Student"
         << setw(5) << "Ave"
         << setw(15) << "Quizzes\n";
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
        { //Display for one st_num:
            cout << setw(10) << st_num
                << setw(5) << st_ave[st_num-1] << " ";
            for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES;
quiz_num++)
                cout << setw(5) << grade[st_num-1][quiz_num-1];
            cout << endl;
        }
}

```

```

    }
    cout << "Quiz averages = ";
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES;
quiz_num++)
        cout << setw(5) << quiz_ave[quiz_num-1];
    cout << endl;
}

```

**Output:**

```

Student Ave    Quizzes
  1 10.0  10  10  10
  2  1.0   2   0   1
  3  7.7   8   6   9
  4  7.3   8   4  10
Quiz averages =  7.0 5.0 7.5

```

## 7.5 Strings

A string is an array of characters terminated by '\0' (null character). There is already a predefined string class in the Standard Library, whose members are declared in the <string> header. To distinguish the strings from the string class, the earlier strings are referred as cstrings. However, the cstring places a special character '\0' immediately after the last character of the string. The declaration of cstring variable is similar to the character array variable.

**Syntax:**      char cstringname[maxsize+1];  
           e.g.:    char name [15];

The "+1" allows space for null character, which terminates any cstring stored in array. The cstring variables can be initialized when it is declared.

```

e.g.: char str[ ]="hello";
      char str1[20]="hello";

```

The above statement initializes the cstring variables. The first statement declares a cstring "str" as a character array of size 6. In the second initialization str, is an array of size 20 and places or stores the value "hello" from 0 to 5<sup>th</sup> index variable including null character ('\0').

---

## 7.5.1 Index Variable Of cstrings

---

As a cstring is an array, it has index variables. These variables are used like those of any other array.

Ex: - `str[ ]="hello";`

This is same as

```
str[0]='h';
str[1]='e';
str[2]='l';
str[3]='l';
str[4]='o';
str[5]='\0';
```

The index variable of the cstring can be manipulated in all possible ways but the null character should be present at the end of the stream.

**Write a program to read a string of characters and display it by adding '2' to each character.**

```
#include<iostream.h>
void main ( )
{
    char s;
    int n;
    cout<<"enter how many characters:";
    cin>>n;
    int i=1;
    while(i<=n)
    {
        cout<<"enter character:";
        cin>>s;
        if(s=='y'||s=='z')
        {
            cout<<char (s-24)<<endl;
        }
        else
        {
            cout<<char (s+2)<<endl;
            i++;
        }
    }
}
```

**output:**

enter how many characters:3

```
enter character:e
g
enter character:t
v
enter character:z
b
```

---

## 7.5.2 Operation On Strings

---

**Reading a cstring:** The value of a cstring variable can read using “cin” with the extraction operator. The extraction operator reads the input stream into the cstream variable until the first blank in the input is accessed.

**e.g.:** - char name [20];  
cin>>name;

In the above statement if we give “JKC college” as input it reads only JKC, as a blank is encountered.

**Reading input including blank space:** A member function getline can be used to read a line of input with blank spaces and place the string of characters into a string variable.

**Syntax:** Inputstreamobject. getline (cstring var,size,delimiting sy)

**e.g.:-** cin.getline (name, 20, '\*');

The string terminates when '\*' is encountered.

**String output:** A string output statement sends the character until it finds the null character. It will not print the character after the null character if there are such characters. If the output statement does not find the null character, it continues to insert the characters into the output stream giving unpredictable output.

**e.g.:**

```
cout<<name;
```

---

## 7.5.3 cstring Predefined Functions

---

The predefined cstring functions are found in <string.h> or <cstring> library file.

**Length of a string:** The length of a string can be found using `strlen()`.

**Syntax:** `strlen(string);`

It returns an integer equal to the length of the source string. It does not count the null character.

e.g.: `y=strlen("hello");`

**Copying a string:** One `cstring` variable can be copied into another `cstring` variable using `strcpy()` function.

**Syntax:**

`Strcpy (target string,source string);`

e.g.: `char x[ ]="hello";`  
`char y[ ];`  
`Strcpy(y,x);`

It copies the source string values into the target string. This function does not check the size of target string variable, if it is large enough to hold the value of source string.

**Note:** A `cstring` variable cannot be used with assignment operators.

**Comparing cstrings:**

Two strings can be compared using `strcmp()` function .

**Syntax:** `strcmp(cstring1,cstring2);`

This function returns 0, if `string1` and `string2` are same. It returns a value less than 0 if `string1` is less than `string2` and greater than 1, if `string1` is greater than `string2`. The strings are compared in lexicographic ordering, where (`a<b<.....<z`).

eg: `char name1[10],name2[10];`  
`cin.getline(name1,10);cin.getline(name2,10);`  
`X=strcmp(name1,name2);`

**Concatenating two strings:** Two strings can be concatenated using `strcat` function.

**Syntax:**

`Stract(targetstring,src_string);`

It is to form by a longer cstring by placing the two shorter cs trings end-to-end. The first argument must be a cstring variable. The second argument can be anything that evaluates to a cstring value, such as quoted string. The result is placed in the cstring variable that is the first argument.

e.g.:

```
char stringvar[20] ="The rain ";  
strcat(stringvar,"in spain");
```

This code will change the value of stringvar to "The rain in Spain" does not check to see that target stringvar is large enough to hold the result of the concatenation.

---

## 7.5.4 Defining cstring Functions

---

A cstring variable is an array, so a cstring parameter to a function is simply an array parameter. The size of the cstring variable should be included, whenever a function changes the value of a string parameter. The null character is used to detect the end of the string value that is stored in the cstring variable.

---

## 7.5.5 Array Of Strings

---

In C++ an array of cstrings is represented as a two dimensional array of characters. For example, the following declares an array called name, which can hold a list of five names, with at most 19 characters with one indexed variable holding '\0' (null character).

```
char name [5][20];
```

An array of cstrings can be manipulated by using both indexes simultaneously, but it is nice to manipulate only one index at a time. A list of cstrings can be manipulated by a loop that steps through values of first index and treats each indexed variable – such as name[0], name[1], name[2] and so forth- as a single cstring variable that can be manipulated by some cstring function.

## //Program to pass cstrings into functions

```
#include<iostream.h>
#include<string.h>
void read(char x[][10],int y);
void write(char x[][10],int y);
void main()
{
char s[5][10];
read(s,5);
write(s,5);
}
void read(char x[][10],int y)
{
for(int i = 0;i<y;i++)
cin.getline(x[i],10);
}
void write(char x[][10],int y)
{
for(int i = 0;i<y;i++)
cout<<x[i]<<"\n";
}
```

### output:

```
hello
hai
bye bye
see you
good day
hello
ha
ibye bye
see you
good day
```

## //Program to sort a given array of strings

```
#include<iostream.h>
#include<string.h>

void read(char x[][10],int y)
{
for(int i = 0;i<y;i++)
cin.getline(x[i],10);
}
void write(char x[][10],int y)
```

```

{
for(int i = 0;i<y;i++)
cout<<x[i]<<"\n";
}
void sort(char x[][10],int y)
{
int i,j;
for(i = 0;i<y-1;i++)
{
for(j=0;j<(y-1-i);j++)
{
if (strcmp(x[j],x[j+1]) >0)
{
char t[10];
strcpy(t,x[j]);
strcpy(x[j],x[j+1]);
strcpy(x[j+1],t);
}
}
}
}
void main()
{
int x=3;
char arr[5][10];
cout<<"Enter the no of strings;";
read(arr,x);
sort(arr,x);
cout<<"Strings after sorting;\n";
write(arr,x);
}

```

**output:**

```

Enter the 3 strings:rama
krishna
govinda

```

```

Strings after sorting:
govinda
krishna
rama

```



## //Program to overload string operations

```
#include<iostream.h>
#include<string.h>
class string
{
char str[40];
public:
string()
{
strcpy(str,'\0');
}
string(char x[])
{
strcpy(str,x);
}
void operator=(string s)
{
strcpy(str,s.str);
}
int operator ==(string s)
{
if((strcmp(str,s.str))!= 0)
return 0;
else
return 1;
}
int operator <(string s)
{
if((strcmp(str,s.str))< 0)
return 1;
else
return 0;
}
friend ostream& operator<<(ostream& out,string s);
friend istream& operator>>(istream& in, string &s);
friend string operator+(string ss1,string ss2);
};
string operator+(string ss1,string ss2)
{
strcat(ss1.str,ss2.str);
return ss1;
}
ostream& operator<<(ostream& out,string s)
{
```

```

out<<s.str;
return out;
}
istream& operator>>(istream& in,string &s)
{
in>>s.str;
return in;
}

void main()
{
string s1,s2("Hello"),s3,s4;
cout<<"Enter two strings:";
cin>>s3>>s4;
s1 = s2+s3+s4;
cout<<"s1 after concatenating s2,s3,s4:"<<s1<<endl;
cout<<"s2"<<s2<<endl;
if (s3 == s4)
cout<<"s3 and s4 r equal\n";
else
cout<<"s3 and s4 are different\n";
if(s3<s4)
cout<<s3<<" is less than "<<s4<<endl;
}

```

**output:**

```

s1 after concatinating s2,s3 and s4:Helloramakrishna
s2:Hello
s3:rama
s4:krishna
s3 and s4 are different

```

---

## 7.6 C++ Standard String Class

---

C++ provides a simple, safe alternative to using chars to handle strings. The C++ string class, part of the std namespace, allows to manipulate Strings safely.

**Declaring a string is easy:**

```

using namespace std;
string my_string;

```

```
or
std::string my_string;
```

An initial value for the string can be specified in a constructor:  
using namespace std;  
string my\_string("starting value");

String I/O is easy, as strings are supported by cin.  
cin>>my\_string;

To read an entire line at a time, the getline function can be used and passed in an input stream object (such as cin, to read from standard input, or a stream associated with a file, to read from a file), the string, and a character on which to terminate input. The following code reads a line from standard input (e.g., the keyboard) until a new line is entered.

```
using namespace std;
getline(cin, my_string, '\n');
```

Strings can also be assigned to each other or appended together using the + operator:

```
string my_string1 = "a string";
string my_string2 = " is this";
string my_string3 = my_string1 + my_string2;
```

```
// Will output "a string is this"
cout>>my_string3>>endl;
Naturally, the += operator is also defined
```

**String Comparisons:** One of the most confusing parts of using char\*s as strings is that comparisons are tricky, requiring a special comparison function, and using tests such as == or < don't mean what you'd expect. Fortunately, for C++ strings, all of the typical relational operators work as expected to compare either C++ strings or a C++ string and either a C string or a static string (i.e., "one in quotes").

For instance, the following code does exactly what you would expect, namely, it determines whether an input string is equal to a fixed string:

```
string passwd;
getline(cin, passwd, '\n');
if(passwd == "xyzy")
{
    cout<<"Access allowed"; } }
```

**String Length and Accessing Individual Elements:** To take the length of a string, you can use either the `length` or `size` function, which are members of the `string` class, and which return the number of characters in a string:

```
string my_string1 = "ten chars.";
int len = my_string1.length(); // or .size();
Strings, like cstrings can be indexed numerically.
```

For instance, you could iterate over all of the characters in a string indexing them by number, as though the string were an array.

Note that the use of the `length()` or `size()` function is important here because C++ strings are not guaranteed to be null-terminated (by a `'\0'`). (In fact, you should be able to store bytes with a value of 0 inside of a C++ string with no adverse effects. In a cstring, this would terminate the string!)

```
int i;
for(i = 0; i < my_string.length(); i++)
{
    cout<<my_string[i];
}
```

Incidentally, C++ string iterators are easily invalidated by operations that change the string, so be wary of using them after calling any string function that may modify the string.

**Searching and Sub strings:** The `string` class supports simple searching and sub string retrieval using the functions `find()`, `rfind()`, and `substr()`. The `find` member function takes a string and a position and begins searching the string from the given position for the first occurrence of the given string. It returns the position of the first occurrence of the string, or a special value, `string::npos`, that indicates that it did not find the sub string.

```
int find(string pattern, int position);
```

This sample code searches for every instance of the string "cat" in a given string and counts the total number of instances:

```
string input;
int i = 0;
int cat_appearances = 0;
```

```

getline(cin, input, '\n');

for(i = input.find("cat", 0); i != string::npos; i = input.find("cat",
i))
{
    cat_appearances++;
    i++; // Move past the last discovered instance to avoid
finding same
        // string
}
cout<<cat_appearances;

```

Similarly, it would be possible to use `rfind` in almost the exact same way, except that searching would begin at the very end of the string, rather than the beginning.

On the other hand, the `substr` function can be used to create a new string consisting only of the slice of the string beginning at a given position and of a particular length:

```

// sample prototype
string substr(int position, int length);
For instance, to extract the first ten characters of a string, you might
use:
string my_string = "abcdefghijklmnop";
string first_ten_of_alphabet = my_string.substr(0, 10);
cout<<The first ten letters of the alphabet are "
    <<first_ten_of_alphabet;

```

### Typical calls to members of the Standard Class `string`:

Members	Remarks
<b>Constructors</b>	
<code>string str;</code>	Default constructor creates empty string objects.
<code>string str("string");</code>	Creates a string object with data "string".
<code>string str("aString");</code>	Creates a string object <code>str</code> that is a copy of a string, which is an object of the class <code>string</code> .

## Elements access:

<code>str[i]</code>	Read/write access to character at index <code>i</code> .
<code>str.substr(position, length)</code>	Returns sub string of calling object starting at position for length characters (read-only access).
<code>str.c_str()</code>	Returns read-only access to the cstring of data in string <code>str</code> .
<code>str.at(i)</code>	Returns read/write reference to character in <code>str</code> at index <code>i</code> .

## Assignment/modifiers:

<code>str1=str2;</code>	Allocates space and initializes it to <code>str2</code> 's data, release memory allocated for <code>str1</code> , sets <code>str1</code> 's size to <code>str2</code> .
<code>str1 +=str2</code>	Character data of <code>str2</code> is concatenated to the end of <code>str1</code> ;the size is set appropriately.
<code>str.empty()</code>	Returns true if <code>str</code> is an empty string, false if it is not empty.
<code>str1+str2</code>	Returns a string that has <code>str2</code> 's data concatenated onto the end of <code>str1</code> 's data.The size is set appropriately.
<code>str.insert(pos, str2)</code>	Inserts <code>str2</code> into <code>str</code> beginning at position <code>pos</code> .
<code>str.remove(pos, len)</code>	Removes sub string of <code>len</code> , starting at position <code>pos</code> .

## Comparisons:

<code>str1==str2</code> <code>str1!=str2</code>	Compare for equality or inequality ;returns a Boolean value.
<code>str1&lt;str2</code> <code>str1&gt;str2</code> <code>str1&lt;=str2</code> <code>str1&gt;=str2</code>	All are lexicographical comparisons

<code>str.find(str1)</code>	Returns index of the first occurrence of <code>str1</code> in <code>str</code> .
<code>str.find(str1,pos)</code>	Returns index of the first occurrence of string <code>str1</code> in <code>str</code> ; the search starts at position <code>pos</code> .
<code>str.find_first_of(str1,pos)</code>	Finds first instance of any character in <code>str1</code> in <code>str</code> , starting the search at position.
<code>Str.find_first_not_of(str1,pos)</code>	Finds first instance of any character not in <code>str1</code> , in <code>str</code> , starting search at position <code>pos</code> .

---

## 7.7 Summary

---

- We have studied the definition of arrays, initializing and passing arrays into functions.
- We have also covered the details about the array of classes and arrays as members of class.
- The initialization and accessing of multidimensional arrays, passing the multidimensional arrays into functions are covered in detail.
- The details regarding the `cstrings`, operations on `cstrings`, predefined `cstring` functions, arrays of strings are covered.
- C++ Standard string class and typical calls to members of the standard class `string` are covered.

---

## 7.8 Technical Terms

---

**Array:** A collection of data elements arranged to be indexed in one or more dimensions. They are stored in contiguous memory.

**Cstring:** Array of characters that end with `'\0'`.

**Multidimensional Arrays:** Arrays with 2 or more dimensions .

---

## 7.9 Model Questions

---

1. What are arrays? Explain them in detail.
2. What are array of objects? How are they defined in C++?
3. What are multidimensional arrays? Explain them.
  
4. What is a cstring? How is it different from character array?
5. Explain some predefined string functions in C++.
6. Explain the passing of arrays into functions.
7. Explain the Standard Class string.

---

## 7.10 References

---

Object-oriented programming with C++,

**by E. Bala Gurusamy.**

Problem solving with C++

**by Walter Savitch**

Mastering C++

**by K.R.Venugopal, RajkumarBuyya,  
T.RaviShankar**

---

---

### AUTHOR

**M. NIRUPAMA BHAT**, MCA., M.Phil.,  
Lecturer,  
Dept. Of Computer Science,  
JKC College,  
GUNTUR.