Notes: Array
Class XI

An array is a collection of data elements of same data type. An array is a collection of variables of the same type that are referenced by a common name. Arrays are very useful in a case where quite many elements of the same types to be stored and processed It is described by a single name and each element of an array is referenced by using array name and its subscript no.

Need of an array

Arrays are very useful in a case where quite many elements of the same types to be stored and processed. For example, to store the marks of 50 students we required 50 variables. But if we use array than we can store all 50 student's mark in one array that have 50 elements. Elements of these arrays will be referred to as arrayname[n], where n is the element number in the array. Writing such a program would not only be simple but also very easy to code and understand.

There are three different types of arrays as following:

One-dimensional array: The array that has only one subscript is known as a one-dimensional array. For example, int marks[50]; The above statement declare array marks has 50 elements, mark[0] to mark[49].

Two-dimensional array: The array that has two subscripts is known as a two-dimensional array. For example, int sales[5][12]; Above statement declare an array sales that has 5 rows and 12 columns.

Multi-dimensional array: The array that has more than one dimension is known as multi-dimensional array. int A[3][3][3]; Above statement declare an A array that have three dimensions.

Declaration of Array

Type arrayName[numberOfElements];

Note:

numberofElements must be an integer constant or a symbolic contact declared using const or #define.

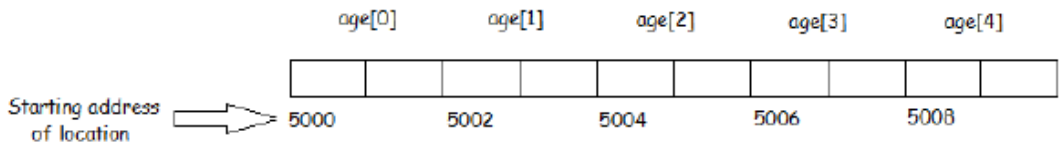
It should be a positive value.

Variables are not allowed.

For example, int Age[5] ;

Suppose an array age of type char with 5 elements declared as int age[5];

Its each element will have sizeof(int) bytes for its storage. On a system with 2 bytes integer size, the array age's element will be having 2 bytes for its storage. If the starting memory location of array age is 5000, then it will be represented in the memory as shown below:



5000 ie. The address of age[0] is also known as base address of the array.

Initialization of One Dimensional Array

An array can be initialized along with declaration. For array initialization it is required to place the elements separated by commas enclosed within braces. int A[5] = {11,2,23,4,15}; It is

possible to leave the array size open. The compiler will count the array size. int B[] = {6,7,8,9,15,12};

Referring to Array Elements

In any point of a program in which an array is visible, we can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value. The format is as simple as: name[index] . The element's number is referred to as an index. An array index in C++ starts with number 0 not 1. That is, array A[5] will be having elements: A[0], A[1], A[2], A[3], A[4].

Examples:

```
cout << age[4]; //print an array element
age[4]=55; // assign value to an array element
cin >> age[4]; //input element 4
```

The elements of an one-dimensional array are stored in contiguous memory location in their index order. For example, an array *grade* of type *char* with 8 elements declared as char grade[8]; will have the element grade[0] at the first allocated memory location, grade[1] at the next contiguous memory location, grade[2] at the next, and so forth.

Memory occupancy of an array

The amount of storage required to hold an array is directly related to its type and size. For a single-dimensional array, the total size (in bytes) can be computed according to following formula: total bytes = sizeof(type) * size_of_array For example, of we have an array age of type int that has 5 elements then the amount of storage required is 2 * 5 = 10.

e.g. If B[10] is a character array, the size of each element is 1 byte. The total bytes required to store B[10] is: 1 x 10 = 10 byrtes , if B[10] is an integer array then each element of B[10] will occupy 2 bytes so 2x10=20 bytes.

Using Loop to input an Array from user

```
int age [10] ;
for (int i=0 ; i<10; i++)
{
cin >> age[i];
}
```

At some moment we may need to pass an array to a function as a parameter. In C++ it is not possible to pass a complete block of memory by value as a parameter to a function, but we are allowed to pass its address. For example, the following function: void print(int A[]) accepts a parameter of type "array of int" called A. In order to pass to this function an array declared as: int arr[20]; we need to write a call like this: print(arr);

Here is a complete example: #include <iostream.h> void print(int A[], int length) { for (int n=0; n<length; n++) cout << A[n] << " "; cout << "\n"; } int main () { int arr[] = {5, 10, 15}; print(arr,3); }

BASIC OPERATION ON ONE DIMENSIONAL ARRAY

Function to Read elements of the array A

```
void Input(int A[], int n)
{
```

```
cout << "Enter the elements:";
for(int i=0;i<n;i++)
cin >> A[i];
}
```
Function to forward traverse the array A
```
void display(int A[], int n)
{
cout << "The elements of the array are:\n";
for(int i=0;i<n;i++)
cout << A[i];
}
```
Function to backward traverse the array A
```
void display(int A[], int n)
{
cout << "The elements of the array are:\n";
for(int i=n-1;i>=0;i--)
cout << A[i];
}
```
Function to Search for an element from A by Linear Search
```
 int Lsearch(int A[], int n, int e)
{
for(int I=0; I<n; I++)
 {
if(A[I]==e)
{
 return 1;
}
}
return 0;}
```

//you accept the array in the main() of the program.
e.g.
Write a program to search an element in an array will be
//linear and binary search
```
#include<iostream.h>
#include<conio.h>
int Lsearch(int A[], int n, int e)
{
for(int I=0; I<n; I++)
 {
if(A[I]==e)
{
 return 1;
}
}
return 0;}

void main()
{
clrscr();
```

You can use a variable say, f and write f++ if you have to count number of times the element is found and returnF at the end instead of return 0;

```
int a[50],i,n,e,ans;
cout<<"Enter the limit"<<endl;
cin>>n;
cout<<endl<<"Enter the elemnts"<<endl;
for(i=0;i<n;i++)
{
cout<<endl;
cin>>a[i];
}
cout<<"\n Enter the element to search";
cin>>e;
result=Lsearch(a,n,e);
if(ans==1)
cout<<"\n Found";
else
cout<<"\n Not Found;
getch();
}
```

It is the call statement to call Lsearch() which will return 1 if found and 0 is not. The result is stored in a variable ans.

Function to Sort the array A by Bubble Sort

```
void BSort(int A[], int n)
{
int I,J,Temp;
for(I=0;I<n-1;I++)
{
for(J=0;J<(n-1-I);J++)
if(A[J]>A[J+1]) {
 Temp=A[J];
A[J]=A[J+1];
A[J+1]=Temp; } } }
```

Function to find maximum and minimum in an array

```
void Maxmin(int arr[],int n)
{
        int min=arr[0],max=arr[0],i;
    for(i=1;i<5;i++)
     {
          if(arr[i]<min)
                min=arr[i];
          if(arr[i]>max)
                max=arr[i];
     }
        Cout<<"\n Maximum value = "<<max<<"\nMinimum value="<<min;
}
```

Function to insert an element in an array

```
void Insert(int a[],int &n,int e)
{
        int i;
    for(i=n;i>1;i)
     {
                a[i]=a[i-1]
```

```
void Insert(int a[],int &n,int l, int e)
{ //when location is passed as parameter
     int i;
      for(i=n;i>1;i)
       {
          a[i]=a[i-1]
       }
     a[1]=e;
     n++; }
```

```
    }
        a[1]=e;
        n++; }
```

Function to delete an element in an array

```
int  Delete(int a[],int &n)
{
        int i;
        int e=a[1];
    for(i=1;i<n-1;i++)
     {
                a[i]=a[i+1]
    }
        n--;
        return e;}
```

```
void Delete(int a[],int &n,int l)
{ //when location is passed as parameter
      int i;
      int e=a[1];
       for(i=1;i<n-1;i++)
        {
            a[i]=a[i+1]
       }
      n--; return e;}
```

Function to find sum of all elements in an array

```
int  sum(int a[],int n)
{
        int i;
        int s=0;
    for(i=0;i<n;i++)
     {
                s+=a[i]
    }
        return s;}
```

```
float Average(int a[],int n)
{ //to find average
      int i,s=0; float avg;
       for(i=0;i<n;i++)
        {
            S+=a[i]
       }
      avg=(float)s/n;
      return avg;}
```

Function to find sum of even and odd elements in an array

```
int  sum(int a[],int n)
{
        int i;
        int sumeven=0,sumodd=0;
    for(i=0;i<n;i++)
     {
                If(a[i]%2==0)
                sumeven+=a[1];
                else
                sumodd+=a[i];
    }
        cout<<"\n Sum of even elements = "<<sumeven
                <<"\n sum of odd elements = "<<sumodd; }
```

Function to replace all the odd vales by its thrice and all the even values by its twice.

```
void Change(int a[],int n)
{
        int i;
    for(i=0;i<n;i++)
     {
```

```
            If(a[i]%2==0)
            a[i]*=2;
            else
            a[i]*=3;
    }
        }
```

Function to reverse an array

```
void Reverse (int a[],int n)
{
        int i,l=n-1,t;
    for(i=0;i<n/2;i++,l--)
     {
            t=a[i];a[i]=a[l];
            a[l]=t;
    }
        }
```

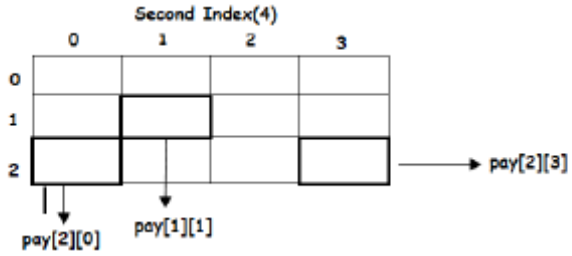
```
void  Reverse(int a[], int n)
{ // Reverse alternate elements
    int i,t;
    for(i=0;i<n-1;i+-2)
     {          t=a[i];a[i]=a[i+1];
                a[i+1]=t;
        }
```

Function to find if an array is palindrome

```
Int Palindrome (int a[],int n)
{
        int i,l=n-1,flag=1;
    for(i=0;i<n/2;i++,l--)
     {
            if(a[i]!=a[l])
                flag=0;
    }
        return flag;
        }
```

Two dimensional arrays

In computing, **row-major order** and **column-major order** describe methods for storing multidimensional arrays in linear memory. Following standard matrix notation, rows are identified by the first index of a two-dimensional array and columns by the second index. Array layout is critical for correctly passing arrays between programs written in different languages. Row-major order is used in C, C++; column-major order is used in Fortran and MATLAB. Two -dimensional arrays are stored in row-column matrix, where the first index indicates the row and the second indicates the column. Suppose we have declared an array pay as follows:
float pay[3][4];
it will be having 3 x 4 = 20 elements which will be represented in memory as shown below:



Row-major order

In row-major storage, a multidimensional array in linear memory is accessed such that rows are stored one after the other. When using row-major order, the difference between addresses of array cells in increasing rows is larger than addresses of cells in increasing columns. For example, consider this 2×3 array:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

An array declared in C as
`int A[2][3] = { {1, 2, 3}, {4, 5, 6} };`
would be laid out contiguously in linear memory as:

```
1 2 3 4 5 6
```

To traverse this array in the order in which it is laid out in memory, one would use the following nested loop:

```
for (i = 0; i < 2; i++)
   for (j = 0; j < 3; j++)
      cout<<A[i][j];
```

Column-major order is a similar method of flattening arrays onto linear memory, but the columns are listed in sequence. The programming languages Fortran, MATLAB, use column-major ordering. The array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

if stored contiguously in linear memory with column-major order would look like the following:

```
1 4 2 5 3 6
```

```
   for (j = 0; j < 3; j++)
      for (i = 0; i < 2; i++)
           cout<<A[i][j];
```

Initialization of Two-Dimensional Array

An two-dimensional array can be initialized along with declaration. For two-dimensional array initialization, elements of each row are enclosed within curly braces and separated by commas. All rows are enclosed within curly braces.

```
int A[4][3] = {{22, 23, 10},
               {15, 25, 13},
```

```
            {20, 74, 67},
            {11, 18, 14}};
```

Referring to Array Elements

To access the elements of a two-dimensional array, we need a pair of indices: one for the row position and one for the column position. The format is as simple as: name[rowIndex][columnIndex]

Examples:

```
cout<<A[1][2];      //print an array element
A[1][2]=13;         // assign value to an array element
cin>>A[1][2];       //input element
```

Using Loop to input an Two-Dimensional Array from user

```
int mat[3][5], row, col ;
for (row = 0; row < 3; row++)
  for (col = 0; col < 5; col++)
    cin >> mat[row][col];
```

Arrays as Parameters

Two-dimensional arrays can be passed as parameters to a function, and they are passed by reference. When declaring a two-dimensional array as a formal parameter, we can omit the size of the first dimension, but not the second; that is, we must specify the number of columns. For example:

```
  void print(int A[][3],int N, int M)
```

In order to pass to this function an array declared as:

```
  int arr[4][3];
```

we need to write a call like this:

```
  print(arr);
```

Here is a complete example:

```
#include <iostream.h>
void print(int A[][3],int N, int M)
{
  for (R = 0; R < N; R++)
    for (C = 0; C < M; C++)
       cout << A[R][C];
}
int main ()
{
  int arr[20][20],m,n,i,j;
  cout<<"\n Enter no of rows";
  cin>>m;
  cout<<"\n Enter no of columns";
```

```
 cin>>n;
 for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
        {
              cout<<"\n enter value in row no"<<i+1<<" column no"<<j+1;
              cin>>a[i][j];
        }
        }
         print(arr,m,n);
         return 0;
        }
```

Function to display content of a two dimensional array A

```
void Display(int A[][20],int N, int M)
{
 for(int R=0;R<N;R++)
 {
   for(int C=0;C<M;C++)
     cout<<setw(10)<<A[R][C];
   cout<<endl;
  }
}
```

Function to find the sum of two dimensional arrays A and B

```
void Addition(int A[][20], int B[][20],int N, int M)
{
 for(int R=0;R<N;R++)
  for(int C=0;C<M;C++)
    C[R][C]=A[R][C]+B[R][C];
}
```

Function to find & display sum of rows & sum of cols. of a 2 dim. array A

```
void SumRowCol(int A[][20], int N, int M)
{                                                    //product of row
 for(int R=0;R<N;R++)
 {
   int SumR=0;
   for(int C=0;C<M;C++)
     SumR+=A[R][C];
   cout<<"Row("<<R<<")="<<SumR<<endl;
  }
```

```
    for(int C=0;C<M;C++)
  {
    int SumC=0;
     for(int R=0;R<N;R++)
      SumC+=A[R][C];
    cout<<"Column("<<C<<")="<<SumC<<endl;
   }
}
```

int Prow=1;

Prow*=A[R][C];

Function to find sum of diagonal elements of a square matrix A

```
void Diagonal(int A[][20], int N)
{
int Rdiag=0,I,j,Ldiag=0;
  for(int I=0;I<N;I++)
   for(j=0;j<N;j++)
           if(i==j)
                Rdiag+=A[I][j];
  for(int I=0;I<N;I++)
   for(j=0;j<N;j++)
            If(Ii+j)==N-1)
    Ldiag+=A[N-I-1][I];
 cout<<"\n Right diagonal"<< Rdiag;
 cout<<"\n Light Diagonal"<<Ldiag;
}
```

int Pcol=1;

cout<<A[i]

will print t

Pcol*=A[R]

Function to find out transpose of a two dimensional array A

```
void Transpose(int A[][20], int B[][20],int N, int M)
{
  for(int R=0;R<N;R++)
    for(int C=0;C<M;C++)
       B[R][C]=A[C][R];
}
Function to Print Triangles
Void Triangle_printing(int A[][20],int N)
{
//upper triangle of left diagonal
cout<<"\n The Upper half of the matrix : \n\n";
 for( i = 0; i < N ; i++)
{
for( j = 0; j < N ; j++)
{
 if(i <= j)
cout << A [i][j] << " " ;
else cout << " " << " ";
```

```
} cout << "\n ";
}
 //lower half of left diagonal -----

cout<<"\n The Lower half of the matrix : \n\n";
for( i = 0; i < N ; i++)
{
 for( j = 0; j < N ; j++)
{ if(i >= j)
cout << A [i][j] << " " ;
else cout << " " << " ";
}
 cout << "\n ";
}
//Upper Triangle of right Diagonal
cout<<"\n The Upper half of the matrix : \n\n";
 for( i = 0; i < N ; i++)
{
for( j = 0; j < N ; j++)
{
 if((i+j) <= N-1)
cout << A [i][j] << " " ;
else cout << " " << " ";
} cout << "\n ";
}
 //lower half of left diagonal -----
cout<<"\n The Lower half of the matrix : \n\n";
for( i = 0; i < N ; i++)
{
 for( j = 0; j < N ; j++)
{ if((i+j)>=N-1)
cout << A [i][j] << " " ;
else cout << " " << " ";
}
 cout << "\n ";  }
```

Function to find the sum of all the elements of two dimensional arrays A

```
void Addition(int A[][20], int N, int M)
{
int sum=0;
float avg;
for(int R=0;R<N;R++)
   for(int C=0;C<M;C++)
      sum+=A[R][C];
avg=(float)sum/(N+M);
cout<<"\n Sum of the elements "<<sum;
```

cout<<”\n Average of all the elements”<<(avg;

}

Strings

```
#include<iostream.h> #include<conio.h> #include<stdio.h>
#include<string.h> void main( ) { char str[50]; int c=0; clrscr();
cout<<"Enter string: "; gets(str); for(int i=0;i<strlen(str);i++)
{ if(str[i]==' ') c++; }cout<<"the no. of spaces: "<<c; getch( ); }
```

String:

This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

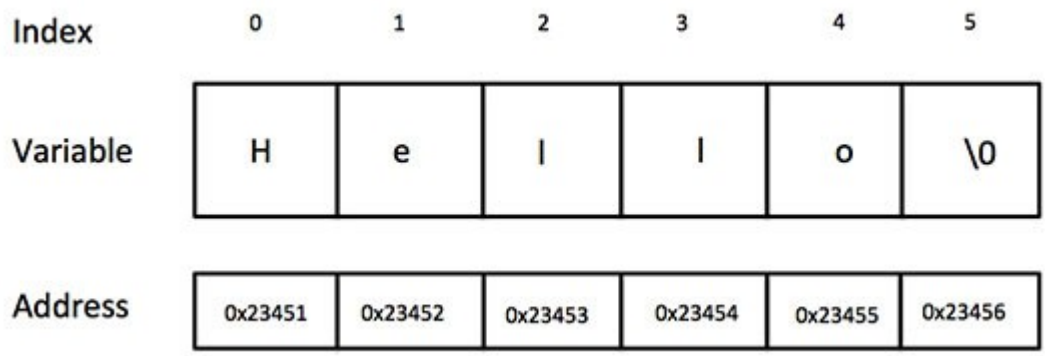
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

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

If you follow the rule of array initialization, then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of above defined string in C/C++:



Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string:

```
#include <iostream>

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

   cout << "Greeting message: ";
   cout << greeting << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Greeting message: Hello
```

Initializing a string

A string can be initialized to a constant value when it is declared.

```
char str[ ] = "Good";
```

Or

```
char str[]={'G','o','o','d','\0'};
```

Here. 'G' will be stored in str[0], 'o' in str[1] and so on.

Note: When the value is assigned to the complete string at once, the computer automatically inserts the NULL character at the end of the string. But, if it is done character by character, then we have to insert it at the end of the string.

Reading strings with/without embedded blanks

To read a string without blanks cin can be used

```
cin>>str;
```

To read a string with blanks cin.getline() or gets() can be used.

```
cin.getline(str,80);
```

-Or-

```
gets(str);
```

Printing strings

cout and puts() can be used to print a string.

```
cout<<str:
```

Or

```
puts(str);
```

Note: For gets( ) and puts(), the header file stdio.h has to be included. puts() can be used to display only strings. It takes a line feed after printing the string.

cin	**gets()**
It can be used to take input of a value of any data type.	It can be used to take input of a string.
It takes the white space i.e. a blank, a tab, or a new line character as a string terminator.	It does not take the white space i.e. a blank, a tab, or a new line character, as a string terminator.
It requires header file iostream.h	It requires the header file stdio.h
Example: char S[80]; cout<<"Enter a string:”; cin>>S;	**Example:** char S[80]; cout<<"Enter a string:"; gets(S);

cout	**puts()**

It can be used to display the value of any data type.	It can be used to display the value of a string.
It does not take a line feed after displaying the string.	It takes a line feed after displaying the string.
It requires the header file iostream.h	It requires the header file stdio.h
Example: char S[80]="Computers"; cout<<S<<S; **Output:** ComputersComputers	**Example:** char S[80]="Computers"; puts(S); puts(S); **Output:** Computers Computers

Counting the number of characters in a string and printing it backwards

```
#include<iostream.h>
#include<stdio.h>
int main( )
{
  char str[80];
  cout<<"Enter a string:";
  gets(str);
  for(int l=0; str[l]!='\0';l++);  //Loop to find length
    cout<<"The length of the string is : "<<l<<endl ;
  for(int i=l-1;i>=0;i--)    //Loop to display the string backwards
    cout<<str[i];
  return 0;
}
```

Function to count the number of words in a string

```
void count(char S[])
{
  int words=0;
  for(int i=0;S[i]!='\0';i++)
  {
    if (S[i]==' ')
      words++;                    //Checking for spaces
  }
```

```
  cout<<"The number of words="<<words+1<<endl;
}
```

Function to find the length of a string

```
int length(char S[ ])
{
  for(int i=0;S[i]!='\0';i++);
     return i;
}
```

Function to copy the contents of string S2 to S1

```
void copy(char S1[ ], char S2[ ])
{
   for(int i=0;S2[i]!='\0';i++)
      S1[i]=S2[i];
   S1[i]='\0';
}
```

Function to concatenate the contents of string S2 to S1

```
void concat(char S1[ ], char S2[ ])
{
   for(int l=0;S1[l]!='\0';l++);
     for(int i=0;S2[i]!='\0';i++)
         S1[l++]=S2[i];
     S1[l]='\0';
}
```

Function to compare strings STR1 to STR2.

The function returns a value>0 if //STR1>STR2, a value<0 if STR1<STR2, and value 0 if STR1=STR2

```
int compare(char STR1[ ],char STR2[])
{
  for(int I=0;STR1[I]==STR2[I] && STR1[I]!='\0'&&STR2[I]!='\0'; I+
+);
     return STR1[I]-STR2[I];
}
```

To reverse the contents of string S and store it in string Rev

```
void Reverse(char S[], char Rev[])
{
   for(int C1=0; S[C1]!='\0'; C1++);
      C1--;
   for(int C2=0;C1>=0;C2++,C1--)
```

```
      Rev[C2]=S[C1];
   Rev[C2]='\0';
}
```

Function to check whether a string S is a palindrome or not

```
int Palin(char S[])
{
   for(int L=0;S[L]!='\0';L++);     //To find length
      for(int C=0;(C<L/2) && (S[C]==S[L-C-1]);C++);
         return (C==L/2)?1:0; //Returns 1 if Palindrome else 0
}
```

Function to change the case of string S to uppercase

```
void Upper(char S[])
{
   for(int i=0;S[i]!='\0';i++)
      S[i] = (S[i]>='a' && S[i]<='z')?(S[i]-32):S[i];
}
```

Function to change the case of string S to lower case

```
void Lower(char S[])
{
   for(int i=0;S[i]!='\0';i++)
       S[i] = (S[i]>='A' && S[i]<='Z')?(S[i]+32):S[i];
}
```

C++ supports a wide range of functions that manipulate null-terminated strings:

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

When the above code is compiled and executed, it produces result something as follows:

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```