**Functions**

It is difficult to maintain and implement a large and complex program with a single list of instruction.. To avoid this problem , a large program is divided into smaller and manageable parts known as modules and this feature is known as Modular programming. In C++ the modularity is achieved through function. **A function is a name given to a group of statements to perform a specific task. .**

**Need for a function are :**

1. Many programs require that a specific operations to be repeated many times. Instead of writing the code many times, we can write it as a single function and access the same function again and again as many times as it is required.

2. It facilitates top down modular programming. In this programming style, the high level operations of the overall problem is solved first while the details of each lower level functions is addressed later.

3. The length of the source program can be reduced by using functions at appropriate places, which saves the memory usage.

4. Programs which use functions are compact & easy to understand. Testing and correcting errors is easy because errors are localized and can be corrected easily. It is easy to locate and isolate a faulty function for further investigation.

5. Functions make a program more readable and understandable to a programmer thereby making the program management and maintenance much easier

6. A single function written in a program can also be used in other programs also. This means that a programmer can build on what others have already done, instead of starting over from scratch.

**Types of functions**

In C++ there are broadly two types of functions

* Library or built-in function – These are available to programmer as a part of the C++ compiler and it can be used in the program by including the corresponding header file. For example pow(), sqrt() present in math.h header file.
* User defined function - These are created by programmer as per the requirement of the program.

**Defining User defined functions**

User defined function is a subprogram defined by the programmer to do a specific operation of any application. Its structure is similar to C++ main function

**Arguments or Parameters**

When a variable is created inside a function, that variable is called **local variable** of that function. Local variable can be referenced (accessed) only within the body of the function where it is declared . Sometime when a function is called by another function, the called function may need some data item (created as local variable of the calling function) for its execution. These data items that the called function must receive for its execution are known as **arguments or parameters**.

According to the method used to pass the arguments to a function definition, the user defined functions can be of two types

- Call by value - Call by reference

**Call by value method**

The call by value method copies the values of parameters (given in the function call statement) into another set of variables created inside the called function i.e the called function creates its own copy of the parameters and then uses them. The argument / parameter refers to the data item that the function must receive when it is executed The parameters which are used in the calling function in the function call statement are termed as **actual parameters** and the copy of the actual parameters used in the called function are known as **formal parameters**. The formal parameters are variables which are given in the function header statement of function definition

**Writing user defined function by call by value method**

Writing a user defined function in a C++ program involves 3 steps:

* Declaration of the function (Function prototype)
* Function Definition
* Calling a function

**Declaration of the function( Function prototype**)

Similar to a variable , a user defined function needs to be declared before they are used in a program. The declaration statement of a function is known as **function prototype** which informs the compiler about the :

* Type of value returned by the function
* The name of the function
* The number and type of arguments/ parameters(if any).

A function prototype is not required when the coding of the function is placed before the calling function. The prototype takes the following form

**Return-type function-name(data type of arguments);**

Where

- Return data type is the type of value to be returned by the function. If no value is returned by the function, the keyword void should be used. If the return data type is omitted, then C++ assumes it to be int.

- Function name is a programmer defined name and should follow the rules of naming any identifier.

- Data type of arguments are the data type of each argument separated by comma. If a function does not require any argument/parameter (i.e. it has an empty argument list), two consecutive () are given after the function name. For example

Example:

void clear(); //prototype of the function clear() which does not have any parameter

float area(float, float); //prototype of a fun. which takes two float parameter and returns //one float data item as return data type.

**Function Definition**

The function definition contains the C++ statements which is responsible for performing the task executed by the function. The function definition consists of

* Function header
* A body

The general form of a function definition is :

**Return-type function-name(argument list) //Function Header statement**

**{**

**<body of the function>**

**}**

The Function Header statement includes the following -:

- **Return data type** which is the type of value to be returned by the function. If no value is returned by the function, the keyword void should be used.

- **Function name** which is a programmer defined name and should follow the rules of naming any identifier.

- The **argument list** which contains name of each formal parameter with its data type .

**Body of the function** contains the statements which are part of the function

**Note :**The function prototype and the function header statement of the function definition must be identical on the return type, function name and the argument/parameter list (number and data type of parameter)

Example 1 : function definition of above given **clear()** prototype

void clear()

{

for(int i=0;i<25;i++)

cout<<’\n’;

}

Example 2: function definition of above given **area()** prototype

float area(float b, float h)

{

return 0.5\*b\*h;

}

**return statement**

The return statement returns the control to the statement immediately following the function call statement. Functions can either return no value or some value of predefined data types.

The return statement can be used to

* Return the calculated result

The function definition of area(..) given in example2 illustrates the use of return statement to return the calculated result to the called function.

* Return the control to calling function in some unusual situations. The following example illustrates this.

Example void squareroot(int a);

void main()

{

int n;

cout<<”enter no whose square root is to be found”;

cin>>n;

squareroot(n);

}

void squareroot(int a)

{

if(a<0)

{

cout<<”root not possible\n”;

return; // returning control (without any value) to called function

}

else

cout<<sqrt(a);

}

Program 1 Write a program to input radius of a circle and calculate circumference, diameter and area of circle. User defined functions should be used to do the calculations.

#include<iostream.h>

#include<conio.h>

float circumference(float);

float diameter(float);

float area(float);

void main()

{

float r,circum;

float dia,ar;

cout<<”Enter radius”;

cin>>r;

circum=circumference( r); //call statement for function circumference

cout<<”circumference”<<circum;

dia=diameter(r ); // call statement for function diameter

cout<<”diameter”<<dia;

ar=area( r); // call statement for function area

cout<<”area”<<ar;

getch();

}

float circumference(float rad)

{

int cir=0;

cir=2 \* rad;

return cir;

}

float diameter(float rad)

{

float d;

d=2 \* 3.14 \* rad;

return d;

}

float area(float rad)

{

float a;

a=3.14 \* rad \* rad;

return a;

}

Program 2 Write a program using user defined function to calculate sum of the series-

-2x2/3! + 4x4/5! -…………2nx 2n/(2n+1)!. The function should accept x and n as parameters and return the sum.

#include<iostream.h>

#include<conio.h>

int fact(int);

int power(int,int);

double sumseries(int,int);

void main()

{

int x1,n1;

double sum=0;

cout<<”Enter value for x and n”;

cin>>x1>>n1;

sum=sumseries(x1,n1);

cout<<”sum of series”<<sum;

getch();

}

double sumseries(int x, int n) //function to calculate sum of the series

{

double s=0;

int i,t=-1;

for(i=2;i<=n;i+=2)

{

s=s + (t\*i\*(double)power(x,i))/fact(i+1); /\*function call statements used in expressions\*/

t=t\*(-1);

}

return s;

}

int power(int x2,int n2) //function to calculate x raised to n

{

int i,p=1;

for(i=1;i<=n2;i++)

p=p\*x2;

return p;

}

int fact(int n3) //function to calculate factorial of a number

{

int i,f=1;

for(i=1;i<=n3;i++)

f=f\*i;

return f;

}

Program 3 Write a menu driven program using user defined functions to do the following operations :

1. Display multiplication table
2. Calculate factorial of a number
3. Calculate divisors of a number
4. Calculate reverse of a number(return type long)
5. To check if a number is prime or not. Return ‘y’ if prime and ‘n’ if not prime.

#include<iostream.h>

#include<conio.h>

void mult(int);

long fact(int);

void divisor(int);

long reverse(int);

char checkprime(int);

void main()

{

int ch,n;

char ans;

do

{

cout<<”Enter the number”;

cin>>n;

cout<<”1: Display multiplication table\n”;

cout<<”2:Calculate factorial of a number\n”;

cout<<”3:Calculate divisors of a number\n”;

cout<<”4: Calculate reverse of a number\n”;

cout<<”5:To check if a number is prime or not\n”;

cout<<”Enter choice”;

cin>>ch;

switch(ch)

{

case 1 : mult(n);

break;

case 2 : long res=fact(n);

cout<<”Result =”<<res;

break;

case 3 : divisor(n);

break;

case 4 : res=reverse(n);

cout<<”reversed no=”<<res;

break;

case 5 : char sol=checkprime(n);

if (sol==’y’)

cout<<”The number is a prime no”;

else

cout<<”The number is not prime”;

break;

}

cout<<”Want to continuey/n”;

ans=getche();

}while(ans==’y’ || ans==’Y’);

getch();

}

void mult(int num) //function to calculate multiplication table of a number

{

int i;

for(i=1;i<=10;i++)

{

cout<<num<<’\*’<<i<<’=’<<num\*i<<’\n’;

}

}

long fact(int num) //function to calculate factorial of a number

{

int i;

long f=1;

for(i=1;i<=num;i++)

f=f\*i;

return f;

}

void divisor(int num) //function to display divisors

{

int i;

cout<<” The divisors are – “:

for(i=2;i<num;i++)

if(num%i==0)

cout<<i<<’ ‘;

}

long reverse(int num) //function to reverse a number

{

long r=0;

int v;

while(num>0)

{

v=num%10;

r=r\*10 + v;

num=num/10;

}

return r;

}

char checkprime(int num) //function to check for prime

{

int i,f=0;

for (i=2;i<num;i++)

{

if(num%i==0)

{

f=1;

break;

}

}

if(f==0)

return ‘y’;

else

return ‘n’;

}

**Limitations of Call by value method**

1. The changes done in the function definition to the formal parameters are not reflected back in the actual parameter created in the calling function.
2. Also the call by value method will be inappropriate in a situation when more than one calculated results need to be returned. This is illustrated in the following example.

Consider the program given below which is used to swap two numbers:

Example : To swap 2 numbers

#include<iostream.h>

void swap(int,int);

void main()

{

int a.b;

a=5,b=10;

swap(a,b);

cout<<a<<b;

}

void swap(int a, int b)

{

int t=a;

a=b;

b=t;

}

Here the output will be **5 10** as the changes done to the formal parameters (a and b) inside the swap() function will not reflect back in the main() i.e the calling function.

The disadvantages of call by value method can be overcome by call by reference method.

**Call by Reference method**

The call by reference method, instead of copying a value of actual parameter into formal parameter, creates an alias of the actual parameter

**Alias**

An **alias** is an alternative name given to the same memory location occupied by an existing variable. This is an alternative name given to a variable which is already created in the memory. Alias is created by using the address of an existing variable using & operator. Hence alias is also referred to as **reference variable**. To create an alias following are the methods:

1. Create the variable
2. Assign the address of variable to the alias by giving the following syntax:

&alias = Name of variable

where &(ampersand) is the address operator

For example int a=5;

int &b=a;

Here **b** is another name given to the memory location of **a** . So here **b** is referred to as **alias** or **reference variable** of **a** .

When a function is called by reference, then the formal parameters become references to the actual parameters in the calling function. This means that in call by reference method, the called function does not create its own copy of original values, rather, it refers to the original values only by different names i.e. the references. Thus unlike call by value method ,in call by reference method, the changes done in the called function are reflected back in the calling function .

Therefore, when a program to swap two numbers is written using call by reference method, the changes in the called function will be done in the original memory location using the alias. Hence the changes done to the variables in swap() will reflect back in main().

Example : To swap 2 numbers using call by reference method

#include<iostream.h>

void swap(int &,int &);

void main()

{

int a,b;

a=5,b=10;

swap(a,b);

cout<<a<<b;

}

void swap(int &a1, int &b1)

{

int t=a1;

a1=b1;

b1=t;

}

The output will be 10 5 . This is because a reference to a i.e. a1 and a reference to b i.e. b1 are being passed to the function.

**Program 2** Write a program to input value in seconds and convert to its equivalent hours, minutes and seconds. The conversion should be done in a user defined function and the calculated results should be displayed in the main() function.

int convert(int &m, int &s);

void main()

{

int h1,m1,s1;

cin>>s1;

h1=convert(m1,s1);

cout<<h1<<m1<<s1;

}

int convert(int &m, int &s)

{

int h;

h=s/3600;

s=s%3600;

m=s/60;

s=s%60;

return h;

}

**Difference between call by value and call by reference method**

**Call by value Call by reference**

1. The call by value method copies the 1. In call by reference method , the

values of actual parameters into formal formal parameters become parameters i.e. the function creates its references to the actual parameters

own copy of arguments values and then i.e the called function refers to the

uses them. original memory location only by different names (alias).

2. The changes done in a separate memory 2. The changes are done in original are not reflected in the original memory location only.

memory location.

3. Example of a Function Prototype: 3. Example of a Function Prototype:

void calc(int); void calc(int &);

**Default Arguments**

C++ allows to assign default value(s) to a function’s formal parameter(s), which is to be used when, required number of argument is not given in the function call statement. In absence of argument, the function picks up the default value of the missing argument. However, when the required number of argument(s) is passed, the actual arguments will overwrite the default argument values.

The default values are assigned to the formal parameters at the time of function declaration...

Example : Illustration of default value in factorial

long factorial(int num=1);

void main()

{

cout<<factorial()<<’\t’; //function will take the default value

cout<<factorial(3); //the passed parameter will overwrite the default value

}

In the above program the output will be 1 6

Example 2:

void calc( int a, int b,int c=4); //function declaration

void main()

{

int a1=2,b1=3,c1=5;

calc(a1,b1); //statement 1

calc(6,7,8 ); //statement 2

}

void calc(int a,int b, int c)

{

int res=a\*b\*c;

cout<<res;

}

In the above function, the function call statement given in statement 1 provides default value 4 to the argument c. However in the function call statement given in the statement 2 ,all three values are being passed in which value 8 is will overwrite the default value of c

Some of the illegal function declarations with default values are:

* void calc(int a=9,int b, int c); //illegal
* void calc(int a=9,int b, int c=7); //illegal

The default value of an argument is used only when it’s matching argument is missing in the function call statement.

**Note :Any argument cannot have a default value unless all arguments appearing on its right have their default values.**

Q 1 Give the output of the following:

#include<iostream.h>

void execute(int &x,int y=200)

{

int temp=x+y;

x+=temp;

if(y != 200)

cout<<temp<<x<<y<<endl;

}

void main()

{

int a=50,b=20;

execute(b);

cout<<a<<b<<endl;

execute(a,b);

cout<<a<<b<<endl;

}

**Output**

50 240

290 340 240

340 240

Q 2 Write the output of the following program:

#include<iostream.h>

int func(int &x,int y=10)

{

if(x%y == 0)return ++x; else return y--;

}

void main()

{

int p=20,q=23;

q=func(p,q);

cout<<p<<q<<endl;

p=func(q);

cout<<p<<q<<endl;

q=func(p);

cout<<p<<q<<endl;

}

**Output**

20 23

10 23

11 11

Q 3 Write the output of the following program:

#include<iostream.h>

int Execute(int M)

{

if(M %3 == 0)

return M\*3;

else

return M+10;

}

void output(int B=2)

{

for(int T=0;T<B;T++)

cout<<Execute(T)<<”\*”;

cout<<endl;

}

void main()

{

output(4);

output();

output(3);

}

**Output**

0\*11\*12\*9\*

0\*11\*

0\*11\*12\*

**Constant arguments**

By constant arguments , it means that these arguments cannot be modified by the function.

Although the constant argument will have features of a constant (i.e. its value cannot be changed), the constant argument cannot be used as the size of an array i.e. in the above example if we try to create an array using b as the dimension/size , it will result in an error.

For example

void calc(const int a, const int b);

void main()

{

int a1=2,b1=5;

calc(a1,b1);

}

void calc(const int a, const int b)

{

int ar[a]; //invalid as an array cannot be declared with a constant inside a function

cout<<a<b; //will print 2 5

a++; //invalid statement as argument a1 cannot be modified

}

**Scope and lifetime of a variable**

**Lifetime of a variable**

Lifetime of a variable refers to the time for which a particular variable remains active in the memory.

**Scope rules/Scope of a variable**

The program parts or range of code of a program in which a particular variable can be accessed is known as **variable scope**. The scope of a variable decides as to how it can be accessed by the different parts of a program.

There are three different scopes:

1. **Local scope** : In C++ a block refers to statements surrounded by a pair of curly braces {..}.Variables declared in a block is local to that particular block {}and can be used only in that block and other blocks contained within it. When the control of program execution reaches the opening braces, these variables come into existence and get destroyed as soon as the control leaves the block through closing brace. The lifetime of the variable is till the execution of the block or block run.
2. **Function scope** : Variables declared inside a function have function scope and are called local variables of that function and can be accessed within the function in which it is declared and gets destroyed when control comes out of the function. Lifetime of the variable is till the function is active or being executed (i.e. function run) and scope is inside the function
3. **File scope** : Variables declared outside all blocks and functions is known as file scope and are referred to as global variables. Since global variables are declared outside all blocks and functions , they are available to all the blocks and functions. Lifetime of the variable is till the program files execution is not over (i.e. program run) and scope is the entire file.

#include<iostream.h>

int x; //global scope

void main()

{

int a; //function scope

:

for(a=1;a<10;a++)

{

int s=a\*a; //Here s has the block scope

cout<<s;

}

:

}

void check(int i) //function scope

{

int temp; //function scope

:

:

}

**Local and Global variable**

A variable which is declared in any function or block are local to that function or block and hence are called **local variable** i.e. these variables can be accessed only in that function. A local variable declared inside a function exists only during the execution of the function. After the execution of a function is over all local variables created in that function get destroyed automatically.

A **Global variable** is the one which is declared outside all the functions and therefore they are available to all the functions in a program. A global variable comes into existence when the program execution starts and is destroyed when the program terminates.

Example :

int g=10; //global variable

void main()

{

int l; //local variable

l=g+10;

cout<<”local=”<<l<<”global=”<<g;

}

A local variable can have the same name as global variable. But in such case, the global variable with the same name remains hidden in the function that declares the local variable. That is the function having a local variable with the same name of a global variable cannot access the global variable directly. In such a case to use the global variable, the global variable should be preceded with **:: (scope resolution operator** )

#include<iostream.h>

int a=20; //global variable

void main()

{

int a=50; //local variable

cout<<::a<<’\n’<<a<<endl;

}

The output would be

20

50

Example 2

#include <iostream.h>

int g=20;

void func(int &x, int y)

{

x=x-y;

y=x\*10;

cout<<x<<’,’<<y<<’\n’;

}

void main()

{

int g=7;

func(g,::g);

cout<<g<<’,’<<::g<<endl;

func(::g,g);

cout<<g<<’,’<<::g<<endl;

}

**Output**

-13,-130

-13,20

33,330

-13,33

Q 1 Given the following code segment:

float a,b;

void main()

{

char ch;

:

:

{

int i=0;

:

}

}

void fn1(char gr)

{

short x,y;

**:**

}

Write scope for the variables given above

**Solution**

- ch has function scope

- i has local scope

- gr has function scope

- x,y have function scope

- a,b file scope

Q 2 Give the output of the following code:

#include<iostream.h>

int m=5;

void main()

{

int m=20;

{

int m=10\*::m;

cout<<”m=”<<m<<”::m=”<<::m<<endl;

}

cout<<”m=”<<m<<”::m=”<<::m<endl;

}

**Output**

m=50 ::m=5

m=20 ::m=5

**Calling functions with array as Parameter**

When an array is given as parameter into a function, the function call will use the array name as parameter. In C++, name of the array represents address of first element ( i.e base address) of array. When an array is used as an argument to a function, only the address of the array gets passed, not a copy of the entire array. Hence array as parameter can only be passed into a function by call by reference method. There are two methods in which an array can be passed as parameter.

**First method** : The receiving parameter of the array may itself be declared as an array, as shown:

#include<iostream.h>

void sort(int a[10]); //prototype

void main()

{

int a[10],i;

cout<<”Enter the array elements”;

for(i=0;i<10;i++)

cin>>a[i];

sort(a);

}

void sort(int a[10]) //function to sort the array elements

{

int i, min, pos, temp,j;

for(i=0;i<9;i++)

{

min=a[i];

pos=i;

for(j=i+1;j<10;j++)

{

if(a[j]<min)

{

min=a[j];

pos=j;

}

}

temp=a[i];

a[i]=a[pos];

a[pos]=temp;

}

for(i=0;i<10;i++)

cout<<a[i];

}

**Second method:** The receiving parameter may be declared as an unsized array

#include<iostream.h>

void sort(int a[]); //prototype

void main()

{

int a[10],i;

cout<<”Enter the array elements”;

for(i=0;i<10;i++)

cin>>a[i];

sort(a);

}

void sort(int a[])

:

:

}

When size of the array is to be inputted from user , it is also to be sent as parameter along with array name

**Example** : Write a program which includes a UDF to search for an integer value from an array of integers using linear search method. The array , the number to search and size should be passed as parameters to the function. The function should return the position of the number in the array if the is found and -1 if the number is not present.

#include<iostream.h>

#include<conio.h>

int search(int a[],int,int);

void main()

{

int a[50],i,n,size,p;

cout<<”Enter the size”;

cin>>size;

cout<<”Enter the array elements”;

for(i=0;i<size;i++)

cin>>a[i];

cout<<”Enter the number to be searched”;

cin>>n;

p=search(a,n,size);

if(p==-1)

cout<<”number not found in the array”;

else

cout<<”number found at”<<p+1<<”position”;

getch();

}

int search(int a[],int n1, int s) //function to search a number in the array

{

int i,pos=-1;

for(i=0;i<s;i++)

{

if(a[i]==n1)

{

pos=i;

break;

}

}

return pos;

}

**Program** Create and input values in arrays for roll number, name of student, marks in five subjects, total, percentage and grade for 5 students. Calculate percentage and assign grade according to a suitable criteria. Write user defined functions for performing the following operations :

* + - Search a roll number and return -1 when not found otherwise return the position. The roll number along with roll number array are to be passed as parameters .
    - Search on name and display the details of the student. The name along with all arrays i.e. rollno, name, marks, perc and grade array are to be passed as parameters .
    - Count number of students of a particular grade when grade is passed as parameter.

The record of each student is to be stored in the corresponding positions of each array.

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

int search\_roll(int rollno[],int rno);

void search\_name(int rollno[],char name[][20],float m1[],float m2[],float m3[],float m4[],float m5[],float perc[],char gd[],char nm[]);

void count\_grade(char gd[],char gr);

void main()

{

int rollno[5],i,pos,rno,ch;

char gd[5],nm[20],name[5][20],gr,ans;

float m1[5],m2[5],m3[5],m4[5],m5[5],tot[5],perc[5];

for(i=0;i<5;i++)

{

cout<<"Enter rollno,name,marks in 5 subjects \n";

cin>>rollno[i];

gets(name[i]);

cin>>m1[i]>>m2[i]>>m3[i]>>m4[i]>>m5[i];

tot[i]=m1[i]+m2[i]+m3[i]+m4[i]+m5[i];

perc[i]=tot[i]/5;

if(perc[i]>=90)

gd[i]='A';

else if(perc[i]<90 && perc[i]>=70)

gd[i]='B';

else if(perc[i]<70 && perc[i]>=50)

gd[i]='C';

else

gd[i]='D';

}

do

{

cout<<"1:Search on rollno \n";

cout<<"2:Search on name\n";

cout<<"3:Count number of students of a particular grade\n";

cout<<"Enter choice";

cin>>ch;

switch(ch)

{

case 1 : cout<<"Enter the rollno to be searched";

cin>>rno;

pos=search\_roll(rollno,rno);

if(pos==-1)

cout<<"roll number not found";

else

cout<<"roll number found at position"<<pos+1;

break;

case 2 : cout<<"Enter the name to be searched";

gets(nm);

search\_name(rollno,name,m1,m2,m3,m4,m5,perc,gd,nm);

break;

case 3 : cout<<"Enter the grade ";

gr=getche();

count\_grade(gd,gr);

break;

default : cout<<"Wrong choice";

}

cout<<"Do you want to continue";

cin>>ans;

}while(ans=='y' || ans== 'Y');

}

int search\_roll(int rollno[],int rno)

{

int i,p=-1;

for(i=0;i<5;i++)

{

if(rno==rollno[i])

{

p=i;

break;

}

}

return p;

}

void count\_grade(char gd[],char gr)

{

int i,cnt=0;

for(i=0;i<5;i++)

{

if(gr==gd[i])

{

cnt++;

}

}

cout<<"Number of students with grade"<<gr<<"is"<<cnt;

}

void search\_name(int rollno[],char name[][20],float m1[],float m2[],float m3[],float m4[],float m5[],float perc[],char gd[],char nm[])

{

int i,pos=-1;

for(i=0;i<5;i++)

{

if(strcmp(nm,name[i])==0)

{

pos=i;

break;

}

}

if(pos==-1)

cout<<"record not found";

else

cout<<"The details are\n";

cout<<"Rollnumber"<<rollno[pos]<<endl;

cout<<"name"<<name[pos]<<endl;

cout<<"marks"<<m1[pos]<<m2[pos]<<m3[pos]<<m4[pos]<<m5[pos]<<endl;

cout<<"percentage"<<perc[pos]<<endl;

cout<<"grade"<<gd[pos]<<endl;

}

In case of two dimensional array, its size must be specified. However the size of first dimension is optional.

**Program1** Write a program to input a two dimensional array and print transpose of it in a function

#include<iostream.h>

void trans(int a[][3]);

void main()

{

int a[3][3],i,j;

for(i=0;i<3;i++0

{

for(j=0;j<3;j++)

{

cout<<”Enter array elements”;

cin>>a[i][j];

}

}

trans(a);

}

void trans(int a[][3])

{

int i,j,b[3][3];

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

b[i][j]=a[j][i];

}

}

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

cout<<b[i][j]<<’ ‘;

}

cout<<endl;

}

}

**Program 2-Write a menu driven program to manipulate a 2-dimensionl numeric array with following choices.**

**- add two matrices";**

**- subtract two matrices";**

**- sum of diagonals of matrix";**

**- row and column total";**

**- Transpose"**

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

void add(int[2][2],int[2][2]);

void subt(int[2][2],int[2][2]);

int adddiog(int[2][2]);

void addrowcol(int[2][2]);

void transp(int[2][2]);

int t,ch,a[2][2],b[2][2],res[2][2],i,j;

char ans;

for (i=0;i<2;i++)

for(j=0;j<2;j++)

{

cout<<"enter element of a and b - ";

cin>>a[i][j]>>b[i][j];

}

do

{

cout<<"\n1. add two matrices";

cout<<"\n2. subtract two matrices";

cout<<"\n3. sum of diagonals of matrix";

cout<<"\n4. row and column total";

cout<<"\n5. Transpose";

cout<<"\nEnter your choice (1-6) - ";

cin>>ch;

switch(ch)

{

case 1: add(a,b);

break;

case 2: subt(a,b);

break;

case 3: int rs=adddiog(a);

cout<<"\nSum of both diagonals of a - "<<rs;

rs=adddiog(b);

cout<<"\nSum of both diagonals of b - "<<rs;

break;

case 4: cout<<"row and column total of a -\n";

addrowcol(a);

cout<<"row and column total of b -\n";

addrowcol(b);

break;

case 5: transp(a);

transp(b);

break;

default:cout<<"invalid choice";

}

cout<<"\nwant to continue(y/n) - ";

cin>>ans;

}while (ans=='y'||ans=='Y');

}

void add(int a[2][2],int b[2][2])

{

int res[2][2];

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

res[i][j]=a[i][j]+b[i][j];

clrscr();

cout<<"Resultant matrix :\n";

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

cout<<res[i][j]<<'\t';

cout<<'\n';

}

}

void subt(int a[2][2],int b[2][2])

{

int res[2][2];

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

res[i][j]=a[i][j]-b[i][j];

clrscr();

cout<<"Resultant matrix :\n";

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

cout<<res[i][j]<<'\t';

cout<<'\n';

}

}

int adddiog(int a[2][2])

{

int d1=0,d2=0;

for(int i=0;i<2;i++)

{

for(int j=0;j<2;j++)

{

if(i==j)

d1+=a[i][j];

if((i+j)==1)

d2+=a[i][j];

}

}

cout<<"\nSum of diagonal 1 - :"<<d1;

cout<<"\nSum of diagonal 2 - :"<<d2;

return (d1+d2);

}

void transp(int a[2][2])

{

int res[2][2];

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

res[i][j]=a[j][i];

clrscr();

cout<<"Resultant matrix :\n";

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

cout<<res[i][j]<<'\t';

cout<<'\n';

}

}

void addrowcol(int a[2][2])

{

for(int i=0,r,c;i<2;i++)

{

r=c=0;

for(int j=0;j<2;j++)

{

r+=a[i][j];

c+=a[j][i];

}

cout<<"\nrow total of row"<<i+1<<" = "<<r;

cout<<"\ncolumn total of column"<<i+1<<" = "<<c;

}

}

Program 3– Write a menu driven program to display length of a string, concatenating 2 strings, reversing a string, coping one string into another

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

# include <string.h>

int len(char s1[80])

{ for(int i=0;s1[i]!='\0';i++);

return i;

}

int cmp(char s1[80],char s2[80])

{

int i=0,res;

while(s1[i]==s2[i])

{

if(s1[i]=='\0' ||s2[i]==’\0’)

{

break;

}

++i;

}

res=s1[i]-s2[i];

return res;

}

void cat(char s1[80],char s2[80])

{

char s3[80];

int k=0;

for(int i=0;s1[i]!='\0';i++)

s3[k++]=s1[i];

for(int j=0;s2[j]!='\0';j++)

s3[k++]=s2[j];

s3[k]='\0';

cout<<”concatenated string is”<<s3;

}

void cpy(char s1[80], char s2[80])

{

for(int i=0;s1[i]!='\0';i++)

s2[i]=s1[i];

s2[i]='\0';

cout<<"\ncopied string - "<<s2;

}

void reverse(char s1[80])

{

for(int i=0, j=strlen(s1)-1,k=strlen(s1)-1;i<k/2;i++,j--)

{

char temp=s1[i];

s1[i]=s1[j];

s1[j]=temp;

}

cout<<"\nthe reverse string is"<<"\n";

puts(s1);

}

void main()

{

clrscr();

char s1[80],s2[80],ans;

cout<<"enter the string1"<<"\n";

gets(s1);

cout<<"enter the string2"<<"\n";

gets(s2);

do

{

cout<<"\n1. Find string length";

cout<<"\n2. Compare strings ";

cout<<"\n3. Concatenate strings ";

cout<<"\n4. Copy one string to other";

cout<<"\n5. reversing of string";

cout<<"\n\nEnter your choice : ";

int ch;

cin>>ch;

if(ch==1)

{

cout<<”length of first string”<<len(s1);

cout<<”length of second string”<<len(s2);

}

else if(ch==2)

{

int res=cmp(s1,s2);

if(res==0)

cout<<"the strings are equal";

else if(res<0)

cout<<"the string1 is smaller";

else

cout<<”string2 is bigger”;

}

else if(ch==3)

{

cat(s1,s2);

}

else if(ch==4)

{

cpy(s1,s2);

}

else if (ch==5)

{

reverse(s1);

}

cout<<"\n\nwant to continue(y/n)";

cin>>ans;

}while(ans=='y' || ans=='Y');

}

**USER DEFINED DATA TYPES**

C++ allows you to define explicitly new data type names using the key word typedef. typedef can be used for fundamental, modified, derived data type.

The user defined data type can be declared using the following

**Syntax: typedef <existing datatype> <new user defined data type>** **;**Example: typedef float money;This statement tells the compiler to recognize money as an alternative name for float. Now a float variable can also be created using money.

Example: Money salary; **typedef does not create any new data type, rather it provides an alternative name for a given data type. .**typedef can also be used to structure i.e. it defines an alias name for the structure tag . Example : typedef struct money

{ int rupees, paisa;} balance;balance salary;

money payment;

Here both money and balance can be used to define the variables of the type of the structure.

**CONSTANTS**

A variable can be defined as constant by using the key word **const** before variable declaration. The syntax to define a constant is:

const datatype variable\_name=value;

const float pi=3.14;

A constant means that its value can never be changed in the program hence should always be initialized a value .Constant variable can only be assigned to other variables.

pi = 6.28 ;// invalid

float val = pi; // valid

const int start // invalid

**#define Preprocessor Directive**

#define directive can also used to define constant and macro.

A constant as explained before when defined cannot change its value throughout the program.

Syntax: #define variablename value

Example : #define PI 3.14

The above statement will translate and replace every occurrence of PI in the program to 3.14.

When #define enclose expression in it, it is called macro.

Example:

#include<iostream.h>

#include<conio.h>

**#define PI 3.14**

**#define area( r ) PI\* r\* r**

void main()

{

float rad;

cout<<”enter radius –“;

cin>>rad;

cout<<”calculate area “<<area(rad);

}

The above program will display the calculated area using the constant and the macro defined using #define preprocessor directive.

**Difference between const and #define preprocessor directive**

**const #define**

1.This can be used to declare array of 1. This cannot be used to define constant array.

Constants

2. This cannot be used to define a 2. This can be used to define a macro

macro

b. #include<iostream.h>

struct PLAY

{ int Score, Bonus ;

} ;

void Calculate (PLAY &P, int N = 10)

{

P.Score++ ; P.Bonus + = N ;

}

void main()

{

PLAY PL = {10, 15};

Calculate(PL, 5) ;

cout « PL.Score « ":"« PL.Bonus « endl ;

Calculate(PL) ;

cout « PL.Score « ":" « PL.Bonus « endl;

Calculate(PL, 15) ;

cout « PL.Score « ":" « PL.Bonus « endl ;

}

**Output**

11 : 20

12 :30  
13 : 45

**Polymorphism**

One of feature of object oriented programming is Polymorphism which means having many forms. In C++ ,this is achieved with the help of function overloading .Function overloading means same function name having several definition that are differentiable by the number and types of their arguments. For example a function add will work in different ways depending on its parameters i.e. when its parameters are 2 integers , it will display the sum and when its parameters are 2 strings , it will display the concatenated string as the result.

The following example illustrates the function overloading

#include<iostream.h>

void add(int,int);

void add(char[],char []);

void main()

{

int ch,a,b;

char s1[10],s2[10];

cout<<”1:Find sum\n”;

cout<<”2:Concatenate strings\n”;

cout<<”Enter choice”;

cin>>ch;

switch(ch)

{

case 1 : cout<<”Enter two numbers”;

cin>>a>>b;

add(a,b);

break;

case 2 : cout<<”Enter two strings”;

gets(s1);

gets(s2);

add(s1,s2);

break;

}

}

void add(int a,int b)

{

int c;

c=a+b;

cout<<c;

}

void add(char s1[],char s2[])

{

char s3[20];

int i,k;

for(i=0;s1[i]!=’\0’;i++)

s3[i]=s1[i];

for(k=0;s1[k]!=’\0’;k++)

s3[i]=s1[k];

s3[i]=’\0’;

cout<<”concatenated string”<<s3;

}

**Example 2** : To calculate area of circle, area of triangle or area of rectangle , the programmer could have given three different function names. But then he needs to remember their names along with related arguments for calculating area of each shape. C++ provides a facility with which the programmer can give same name say calc\_area to more than one functions but each having its own parameters. This way the programmer can concentrate more on logic of program rather than remembering the different function names.

The program for calculating area of different shapes would be:

#include<iostream.h>

double calc\_area(int rad);

double cal\_area(float len, double wid);

double calc\_area(int base, int height);

void main()

{

int ch,r,b,h;

float l;

double w,a;

cout<<"1:calculate area of circle\n";

cout<<"2:calculate area of rectangle\n";

cout<<"3:calculate area of triangle\n";

cout<<"Enter choice";

cin>>ch;

switch(ch)

{

case 1 : cin>>r;

a=calc\_area(r);

cout<<"area of circle"<<a;

break;

case 2 : cin>>l>>w;

a=calc\_area(l,w);

cout<<"area of rectangle"<<a;

break;

case 3 : cin>>b>>h;

a=calc\_area(b,h);

cout<<"area of triangle"<<a;

break;

default : cout<<"wrong choice";

}

}

double calc\_area(int rad)

{

double area;

area= 3.14\*rad\*rad;

return area;

}

double calc\_area(float len,double wid)

{

double area;

area= len\*wid;

return area;

}

double calc\_area(int base,int height)

{

double area;

area= 0.5\*base\*height;

return area;

}

**Note:**

* + - Sometimes when overloaded functions are called ,they might cause ambiguity error. It occurs when compiler may not be able to decide which version of overloaded function should be called.

Example :

long perform(int); //prototype of first function

long perform(long); //prototype of second function

In the above case , since promotion from one data type(int) to (long) , is automatically done by the compiler, the compiler will not be able to differentiate the two functions and hence this will cause ambiguity.

- A function with default arguments should not be given with a function with no arguments, since it would confuse the compiler as to which function to call

For example:

void disp(int c=9);

void disp();

void main()

{

int a=7;

disp();

)

void disp()

{

cout<<”display message”;

}   
 void disp(int c)

{

cout<<”display c value”;

}   
 The above program will generate an error as with the function call statement **disp(),** the compiler will know as to which function to be executed.

**Practical Questions**

1. Write a menu driven program using user defined functions to do the following operations :

a. Generate Fibonacci series where first two numbers and n are taken as parameters .

b. Generate tribonacci series where first three numbers and n are taken as parameters .

2. Write a menu driven program to accept two integers a and b and do the following options. Write UDF for each option.

- count and return number of all prime numbers between a and b, when a and b are given as parameters

- count and return number of all even numbers between a and b, when a and b are given as parameters

3. Write a program using user defined function to calculate sum of the series-

-3!x2 + 5!x4 -…………(2n+1)!x 2n. The function should accept x and n as parameters and return the sum.

4. Write a menu driven program using user defined functions to manipulate a 2-dimensional numeric array with following choices.

- Add two matrices. The function should take two-dimensional arrays as parameters.

- Print the sum of upper triangular matrix elements . The function should take one two-dimensional array as parameter.

- Print Transpose of a matrix. The function should take one two-dimensional array as parameter.

5. Write a menu driven program using user defined functions to manipulate a 1-dimensional numeric array with following choices.

- Find the greatest value from the numbers in the array. The function should take one-dimensional array as parameter and return the maximum value.

- Sort the elements of the array in ascending order. The function should take one two-dimensional array as parameter.

- Reverse the array elements. The function should take one one-dimensional array as parameter.

6. Write a menu driven program to using user defined functions to manipulate a 1-dimensional character array with following choices:

- copy one string to another

- concatenating 2 strings

- check if the string is a palindrome or not

The corresponding functions should take string/strings as parameter.

7. Write program using function overloading to perform the following operations:

- count and return number of words in a string when string is passed as parameter

- count and return number of digits in an integer. The number is passed as parameters

- count number of prime numbers in an integer array and return the count.

8. Define a structure with itemno, name, price and category and date\_of\_purchase (a structure variable with date, month and year). Declare an array of size 10 and do the following operations. Write UDFs for each option.

- Display all item details of a given category. The function should take the structure array and its size and a category as parameter

- Arrange it in ascending order according to price. The function should take the structure array and its size as parameter

* Count the no. of items whose price is more than 500. The function should take the structure array and its size as parameter
* Count the no. of items which are purchased on a particular date. The function should take the structure array and its size and a date as parameter

9. Create a structure containing students data as rno, name, dob, stream, marks in three subjects ,where marks (is an array of size 3) and dob is a structure having date, month and year. Make an array for 10 students. Write a function to calculate the average marks for each student and print the student details in report card form. The function should take the array as parameter and should return the no. of students who have failed i.e. average < 40%.

10. Write a function to add two complex numbers . The function should take two complex numbers as parameter and should return the calculated result. Make use of structure to define the complex number(a number having real part and an imaginary part)