```
#include <stdio.h>
#include <conio.h>
void main()
{
  int time;
  clrscr();
  printf("Enter the time");
  scanf("%d", &time);
  if((time >= 0) && (time <= 12))
    printf("Good Morning");
  if((time > 12) && (time <= 18))
    printf("Good Afternoon");
  if((time > 18) && (time <= 24))
    printf("Good Evening");
  else
    printf("time is out of range");
  getch();
}
```

12-01-19

Looping:

Suppose we want to display "Hello" on output screen 5 ~~ty~~ times in 5 different lines, we might think of writing printf statement 5 times consisting of a string "Hello\n" 5 times.

If we want to like to print hello 500 times the above procedure is quiet lengthy. In C programming there is a facility available to repeat certain works in the form of looping statement.

There are 3 types of loop statement:

1) while - loop
2) do-while loop
3) for loop.

★ The while statement:

It is used to carry out looping operations, in which group of statements executed reapeatedly until some condition has been satisfied.

The general form of statement is

```
while (expression)
{
  statement 1;
  statement 2;
}
```

The statement will be executed repeatedly as long as the expression is true. (non-zero value).

This statement can be simple or compound.

Eg:

```
count = 1;  → /* Initialization of loop counter */
while (count <= 5)  ← condition
{
  printf("Hello\n");
  count = count + 1;  /* increment statement */
}
```

Output:

Hello
Hello
Hello
Hello
Hello
Hello

1 to 100
print even nos from 100 to 200.

PAGE NO:
DATE: / /

Eg: /* Write a program to print 1 to 10 nos.
#include <stdio.h>
#include <conio.h>
void main ()
{
int i;
i=1;
while (i<=10)
{
printf ("\n%d,"i);
i++;
}
printf ("\n program over")
getch ();
}

Eg: /* Write a program to print 1 to 100 nos.
#include <stdio.h>
#include <conio.h>
void main ()
{
int i;
i=1;
while (i<=100)
{
printf ("\n%d,"i);
i++;
}
printf ("\n program over")
getch();
}

PAGE NO:
DATE: / /

Eg: /* Write a program to print even nos from 100 to 200.
#include <stdio.h>
#include <conio.h>
void main ()
{
int i;
i=100;
while (i<=200)
{
printf ("\n%d," i);
i+2;
}
printf ("\n program over")
getch ();
}

17-01-19

* The do-while statement:-

When a loop is constructed using the while statement the taste for continuation of the loop is carried out at the beginning of each pass. Sometimes it is desirable to have a loop with the taste for continuation at the end of each pass.

This can be accomplished by means of the do-while statement.

The general form of do-while statement:

do
{

PAGE NO.:
DATE: / /

```
do
{
  Statements.
}
while (expression);
```

The statement will be executed repeatedly as long as the value of expression is true. Notice that # statement will always be executed atleast once even though the condition or expression has false value at the first time.

Q. Write down the difference b/w while loop & do-while loop with its syntax & general form.

* Write a program using do-while statement to display the integer from 1 to 10

```
# include <stdio.h>
# include <conio.h>
  void main ()
  {
    int integer = 1;
    do
    {
    printf ("\n Integer from 1 to 10");
    printf ("%d", integer);
     integer++;
    }
    while (integer <= 10);
    printf (" program is over");
    getch();
  }
```

PAGE NO.:
DATE: / /

Output:

Integers from 1 to 10
1 2 3 4 5 6 7 8 9 10
program is over.

Modified above program:

```
void
 main ()
{
  int integer = 1;
  printf ("\n integer from 1 to 10");
  do
  {
  printf ("\n %d", integer++);
  }
  while (integer <= 10);
  printf ("\n program is over");
  getch ();
}
```

Output:

Integer from 1 to 10
1
2
3
4
5
6
7
8
9
10
program is over.

More looping with for- statement:

for- Statement is the 3rd Sp most commonly used looping statement in C. This statement includes an expression that specifies an initial value of index (expr. 1), another expression that determines wheather or not loop is continued (expression 2) & the 3rd expression that allows to modify index at the end of each pass (expression 3).

Expression one (1) is executed only once at the beginning of loop, expression 2 is evaluated at the beginning of each pass to the loop, & expression 3 is evaluated at the end of each pass.

18-01-19

* /*Write a program to display odd number from 1 to 100 */

```
# include <stdio.h>
# include <conio.h>
void main ()
{
int i;
clrscr();
printf("add numbers from 1 to 100");
for (i=1; i<=100 ; i=i+2)
{
printf ("\n %d", i);
}
getch ();
}
```

* /* modify the program to display even number */

```
# include <studio.h>
# include < conio.h>
void main ()
{
Int i;
clrscr ();
printf (" even numbers from 1 to 100");
for (i=0; i<=100 ; i=i+2)
{
printf (" \n %d", i);
}
getch ();
}
```

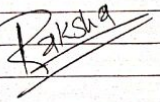
* /* Write a program to display no.s divisible by 5 from 1 to 100 */

```
# include <studio.h>
# include <conio.h>
void main ()
{
int n;
clrscr ();
for (n=1 ; n<=100 ; n++)
{
if (n% 5==0)
printf ("\n %d is divisible by 5", n);
else
printf (" \n %d is not divisible by 5", n);
}
getch ();
}
```

PAGE NO.:
DATE: / /

* /* Write a program to check whether the persons are voting in the list of 50 person the eligibility age is 18 or more than 18 years */

```
# include <stdio.h>
# include <conio.h>
  void main
{
  int p,a ;
  clrscr();
  for (p=1 ; p<50 ; p++)
  {
  printf ("Enter the age");
  scanf ("%d", &a);
  if (a>=18)
  printf ("\n %d the person is eligible for voting", a);
  else
  printf ("\n %d the person is not eligible
          for voting", a);
  }
getch();
}
```



PAGE NO.:
DATE: / /

* Convert the above program to use while loop.

```
# include <stdio.h>
# include <conio.h>
void main()
{
  int p,a;
  while (a<=50)
  {
  printf ("\n Enter the age");
  scanf ("%d", &a);
  of (a>=18)
  printf ("\n %d years the person is eligible
          for voting", a);
  else
  printf ("\n %d years the person is not
          eligible for voting", a);
  }
  getch();
}
```

PAGE NO.:
DATE: 22/01/19

Ch-3.

Functions in C.

* Introduction:

A large program can be broken down into no of smaller, self content component, each of which has some unique & identifiable purpose and this can be acheive by use of function

* What is function:

Thus function is self content program segment that carriers out some specific well define task. The use of function avoids need for repeated programming of the same instructions. The use of function also enables programmer to build customize library of frequently use rutines. Hence a single function can be utilize by many different programs & this avoids repeative programming between program.

Every C program consist of one or more function one of these function must be called main execuation of program will be always carrying out by the instruction in main. Addition function will be available in main function.

If a program contain multiple function their defination appear in any order, though they must be independent on one another, i.e, one function def^n cannot be embedded in the def^n of another function.

A function will carry out its action or task, whenever it is accessed (whenever the function is called) from some portion of the program the same function can be access from several different places within a program once the function has carried out its action, control will be return to the point from which the function was access.

A function will process information i.e, past to it from the calling portion of the program & return a single value.

Information is to past to way function via special identifiers called argument or parameters & return a value via return statement.

The following program shown the use of function in the program

```
/* Calculate a cube of number */
#include <stdio.h>
#include <conio.h>
int cube (int a)        /* fun^n defination */
{
  q = a*a*a;            /* body of function */
  return (q);
}
void main ()
{
  int x, y;
  printf ("Enter number");
  scanf ("%d", &x);
  y = cube (x);
  printf ("\n Cube = ", y);
}
getch();
}
```

PAGE NO :
DATE : / /

23-01-19

★ Defining a function :

A function defination has two principle components the first line (including the argument declaration) & the body of the function. The first of function defination contains the types specification of the value returned by the function followed by function name & set of arguments seperated by comma (,) enclosed in parenthesis (optional).

Each argument is precided by its associated type declaration. The general format of function defination is :

```
return data type fun^n name (argument list)
{                                  first line (or) header line
   =// body of function
}
```

The argument are called formal arguments because they represent the name of data items that are transfer into the function from the calling program are called actual argument or actual parameters.

★
```
# include <stdio.h>
# include <conio.h>
void main ()
{
  int x;
  x = 5;
  y = cube (x);        /* function called */
}
```
→ actual argument

PAGE NO :
DATE : / /

```
/* function defination */
return type — int cube (int a);    /* Header line */
                 ↓ function name      ↳ formal argument
{
  int q;
  q = a x a x a;
  return (q)
}
```

★ Write a program to using the function to find the maximum of two numbers.

```
# include <stdio.h>
# include <conio.h>
void main ()
{
  int x, y;
  clrscr ();
  printf ("\n Enter two number");
  scanf ("%d %d", &x, &y);
  maximum (x, y);
  getch ();
}
  maximum (int a, int b)
{
  int c;
  if (a>b)
    c=a;
  else
    c=b;
  print ("maximum no is = %d", c);
}
```

PAGE NO.
DATE : / /

Pii)
```
#include <stdio.h>
#include <conio.h>
void main()
{
int x, y, z;
clrscr();
printf("\n Enter two number");
scanf("%d %d", &x, &y);
z = maximum(x, y);
printf("maximum = %d", z);
getch();
}
int maximum(int a, int b)
{
int c;
if (a>b)
  c=a;
else
  c=b;
return(c);
}
```

/* Write a program to find factorial of number using function */

```
#include <stdio.h>
#include <conio.h>
void main()
{
long int f;
int x;
scanf("%d", &x);
f = factorial(x);
printf("factorial = %d", f);
getch();
}
long int factorial(int a)
{
int i;
long int p=1;
for(i=1; i<=a; i++)
{
p= p*i;
}
return(p);
}
```

29-01-19

Accessing the function:

A function can be access i.e, called by specifing its name, followed by list of arguments enclose in parenthesis & seperated by comma & end with semicolon.

A function called may be a part of simple expression (such as assignment statement), or may be one of the operatant within a more complex statement.

The arguments appear in the function called are referred as actual argument where as arguments appear in the first line of function defination are called as formal argument.

The actual argument must be of the same data type as its corresponding formal argument & the no of argument in actual must be equal to formal

argument.

```
a = function name (argument list)
{
printf (" cube = %d", cube (5));
  x = 10;
  y = cube (x+5);
printf (" cube = %d", y);

  z = max (x, y);
  y = cube (x);
  y = cube (5);
  y = cube (10+x);
```

In the above example the function access returned by function to be assign to the variable y here the arguments are constant, variables & expressions also (for ex, 5, x, 10+x)

```
Display (a, b, c);
```

This function access the values of a, b & c to be process internally (i.e, display) within the func.

To declear the function for a program is called function prototype.

Function prototype are usually written at the beginning of program i.e, before main the general form of function prototype is.

datatype function name (type argument list, type arguments).

Where data type represent the data type of item i.e, written by the function. function name is any valid identifier & type is the data type of each argument in the list.

* The following program illustrate the function prototype, function access, function defn in the program.

```
# include <stdio.h>
# include <conio.h>
int max (int x, int y);   /* function prototype */
void main ()
{
 int a, b, c;
printf ("\n Enter two nos:");
scanf ("%d %d", &a, &b);
  c = max (a, b);
  printf ("\n maximum no. = %d", c);
getch ();
}
 int max (int x, int y)
{
  int z;
  if (x > y)
    z = x;
  else
    z = y;
  return (z);
}
```

PAGE NO:
DATE: / /

31-01-19

* Passing Argument to function :

When a single value is passed to a function via of actual argument the value of corresponding formal argument can be altered within the function but the value of actual argument within the calling routing will not change. This procedure called passing the value for an argument to a function is known as passing by value.

Passing an argument by value has advantages & disadvantages. If the actual argument is expressed simply as a single variable, it protects the value of this variable from alteration within the function on the other hand it does not allow information to be transferred back to the calling portion of the program via arguments.

Thus passing by value is restricted to one way transfer of information where as passing by reference helps to transfer of information in two ways.

* Write a program passing argument by value passing arguments to function.

```
# include <stdio.h>
# include <conio.h>
void modify (int x, int y); /* fun^n prototypes */
void main ()
{
  int a, b;
  printf ("enter value for a, b");
  scanf ("%d, %d", &a, &b);
  printf ("\n value before function calls %d %d", a, b);
  modify (a, b);
  printf ("\n value after function %d %d", a, b);
  getch ();
}
void modify (int x, int y)
{
  x = x+5;
  y = y+5;
  printf ("\n value after modify in fu^n %d %d", x, y);
}
```

output:

```
Enter value for a, b    10  20
value before fun^n      10  20
value after modify in fun^n   15  25
value after fu^n call   10  20
```

* /* modify the above program for passing argument by reference */

```
# include <stdio.h>
# include <conio.h>
void modify (int * x, int *y); /* fu^n prototype */
void main ()
{
  int a, b;
  printf ("Enter value for a, b");
  scanf ("%d %d", &a, &b);
```

```
printf ("\n value before fun^n call %od %od", a,b);
modify (&a, &b); /* calling fu^n with reference or passing
                    argument as reference */
printf ("\n value after call %d %d", a,b);
getch ();
}
void modify (int *x, int *y).
{
  x = *x + 5;
  y = *y + 5;
printf ("\n value after modify fun^n %d %d", x,y);
}
```

output:

```
Enter value for a b    10  20
value before fun^n call    10  20
value after modify in fun^n    15  25
value after fun^n call    25  25
```

★ Recursion:

Recursion is a process by which function is called it self repeatedly until some specified condition, has been satisfied. The process is use for repeatative computation in which each action is stated in terms of previous result.

To solve problem recursively to condition must be satisfied.

i) The problem must be return in recursive form.
ii) Problem statement must include stopping condition.

The following program illustrate this

```
#include <stdio.h>
#include <conio.h>
long int factorial (int x); /* fun^n declaration */
void main ()
{
  int x;
  scanf ("%d", &x);
  printf ("\n factorial of no. = %d", factorial (x));
  getch ();
}
long int factorial (int n)
{
  if (n <= 1)
  return (1)
  else
  return (n * factorial (n-1))  /* calling factorial fu^n */
}
```

Q. What is function? what are use of function.
Q. Explain two types of arguments & the purpose of return statement.
Q. What is function prototype? Write a difference b/w call by value & call by reference.