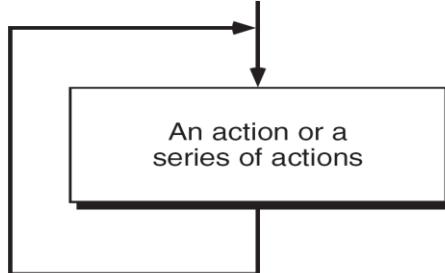


Repetition/Iteration(Looping)

The ability of computers to repeat/iterate an operation or a series of operations many times is called looping. **The construct that supports looping is called “loop”.**



Concept of loop

Loops have 3 functional parts-

- *Loop control expression (conditional statement(testing statement))
- *Loop body (1 or more statements)
- *Loop updation (manipulation) statement

Loop control Expression:-

It is a test to be performed either before or after each iteration/repetition (on LCV-loop control variable)so that the loop can be checked (controlled) for its start/ no. of times repetitions and end operations.

- Before the loop starts we perform initialization(of LCV), process in 2 ways:
 - Explicit – we give any specific start value to the loop control variable in code.
 - Implicit – any preexisting situation gives it.

Body of the loop:

It has one more processes (actions) defined in the form of various statements. These statements are executed only if test-condition on LCV becomes true. It has last statement -updation (of LCV)in general along with other actions.

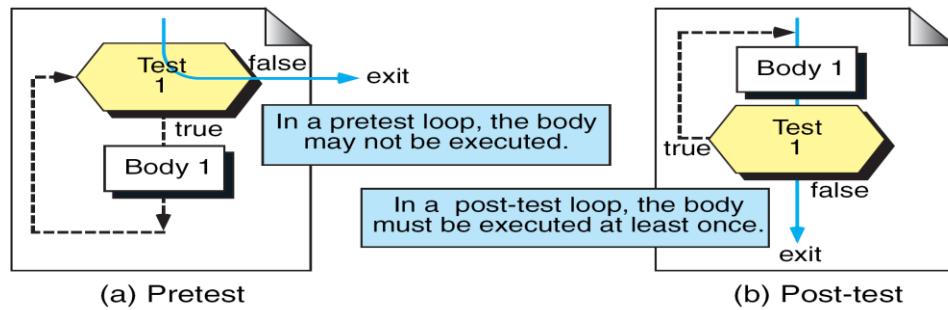
Updation :- Under this process the value of the variable(loop control variable(s)) being tested in the condition may change. Updation may be **+ve/-ve** (increment/decrement) so loops may go in forward/backward directions.

Updation is done in each repetition as a last action (statement) of the loop body. After few updations LCV terminates the loop (when result of condition on LCV becomes false). If loop body is repeated :- m times, then updation is also done :- m times & **testing will be done: m+1 times**

Types of loops :- (on the basis of place of test & control)

- **Entry controlled loops (pre-test loops)**
The control expression is tested even for the first time to start execution of loop-body, and tested again for each iteration(repetition), if test results as true, the loop continues (body executed); otherwise, the loop is terminated.
- **Exit controlled loops (post-test loops)**
In first iteration, the loop (body) is executed. Then the control expression is tested. If it is true, a next iteration is started; otherwise, the loop terminates.

A program loop $\xrightarrow{\text{is made of}}$ body of the loop + control expression

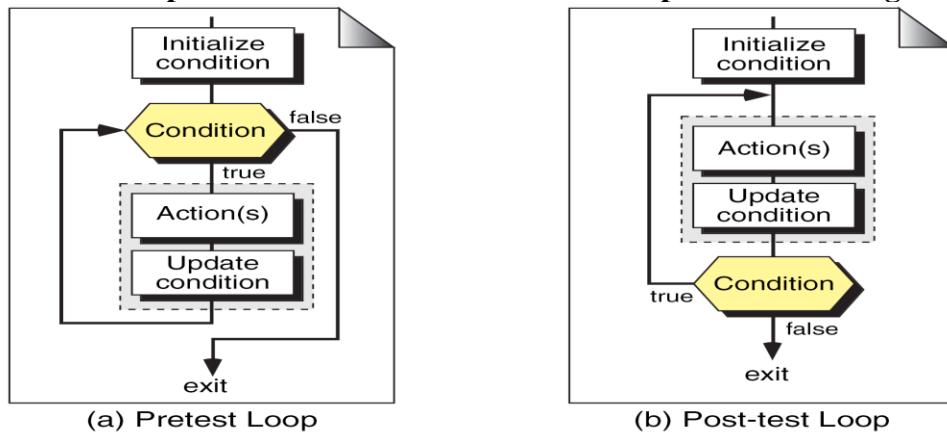


Body of the loop \equiv An action or series of actions to be done.

Another type of loops:

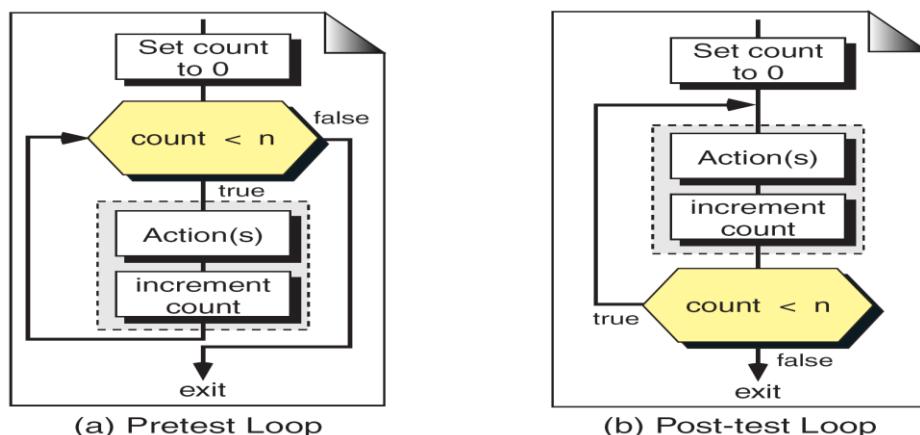
1-Event-controlled (sentinel loops) – An event or sentinel value changes the control expression from true to false. eg. End of file,end of data $\equiv -1$. value. While & do-while are generally used here

No. of repetitions is not known before the loop starts executing.



Event-controlled Loop Concept applied in both types of loops

Counter-controlled loops – when we already know the no. of times the actions are to be repeated, we use this type of loop. Here we use a control variable – counter/count. Counter must be initialized, updated (+ve or -ve) & tested for desired loop iterations. For loop is used here.



Counter-controlled Loop Concept applied in both types of loops

Loops in C

3 loop statements are defined in C-

- o **while** - pretest loop (best for event-controlled loops)
- o **for** - pretest loop (best for counter-controlled loops)
- o **do...while** - post-test loop(best for event-controlled loops)

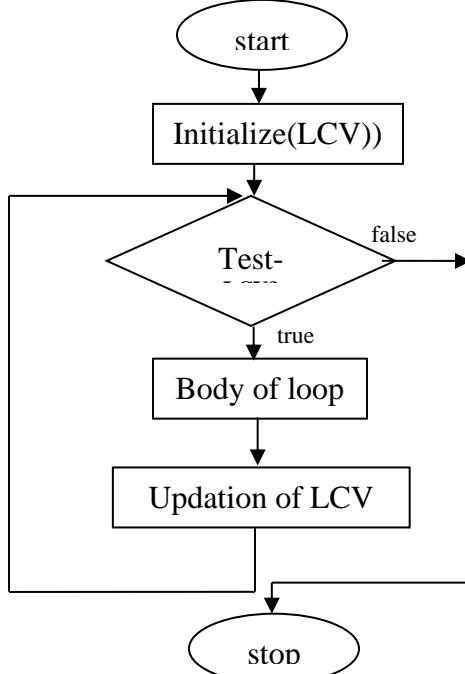
While loop :- It is a pretest loop so it checks each time the expression (condition) before it executes then repeats.

Initialize LCV

while(test cond.on LCV ≡ expression)

{ Body of loop ≡ many actions
 Update-LCV(loop control variable)
}

Flow-chart: Operations of while loop:



Ex.: Calculation of sum of nos. 1 to 10:-

```
void main()
{
    int num, sum;
    num=1;sum=0;
    while (num <11)
    {
        sum = sum+num;
        num++;
    }
    Printf("sum is=%d", sum);
}
```

- While (condition); - indefinite loop(runs infinite times due to absence of updation of LCV).
 - Use and observe the difference in: while (`i++ < 10`) & While (`++i < 10`)
- while (true) → indefinite loop (process control loop)
- while (`scanf("%d", &x) !=EOF`)
 - in Dos ctrl + d → denotes ends of file.

Examples: Programs using while loop:

=>WAP to calculate value of x^y ('x' to the power 'y') where x and y are entered by keyboard.

Logic :1

```
main()
{
    int x,y,z=1,i=1;
    printf("enter the number and its
power ");
    scanf("%d%d",&x,&y);
    while(i<=y)
    {
        z=z*x;
        i++;
    }
    printf("%d",z);
}
```

Logic :2

```
main()
{
    int x,y,z,i=1;
    printf("enter the number and its power ");
    scanf("%d%d",&x,&y);
    z=x;
    while(i<y)
    {
        z=z*x; i++;
    }
    printf("%d",z);}
```

=>WAP to calculate factorial x! of a given x.

Solution1:Using Incremented loop

```
main()
{
int z,x,i=1;
printf("enter a number to get its
factorial\n");
scanf("%d",&x);
z=x;
while(i<z)
{
    x=x*i;
    i++;
}
printf("\nfactorial is =%d",x);
}
```

Solution2: Using Decremented loop

```
main()
{
int x,i;
printf("enter a number to get its factorial\n");
scanf("%d",&x);
i=x-1;
while(i>0)
{
    x=x*i;
    i--;
}
printf("\nfactorial is =%d",x);
}
```

WAP to reverse the digits of given number

```
main()
{
    int num,rev=0,rem;
    printf("enter number to reverse its digits");
    scanf("%d",&num);
    while(num!=0)
    {
        rem=num%10;
```

WAP to sum the digits of given number

```
main()
{
    int num,sum=0,rem;
    printf("enter a number to sum its digits");
    scanf("%d",&num);
    while(num>0)
    {
        rem=num%10;
```

```

rev=rev*10+rem;
num=num/10; }
printf("%d",rev);  }

```

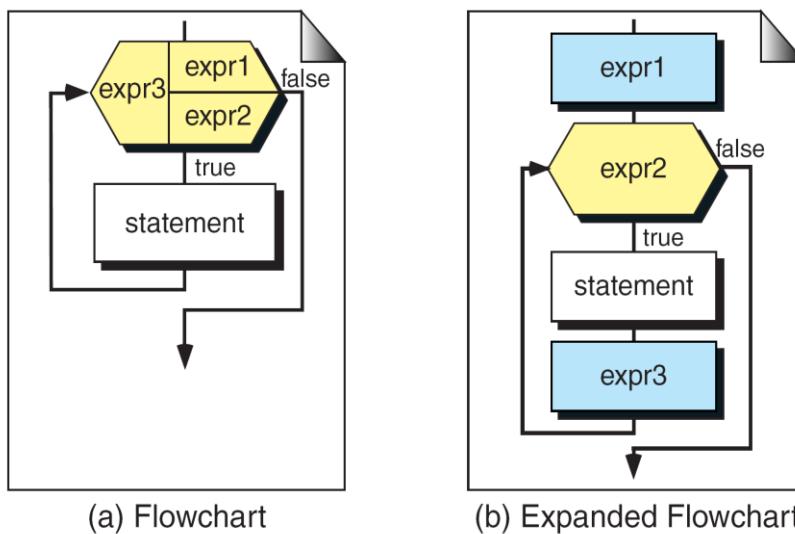
```

sum +=rem;
num=num/10; }
printf("%d",sum);  }

```

The For loop :-

- It is a pretest loop that uses ‘3’ expressions: [For(**Exp1;Exp2;Exp3**){body of loop}]
 Expression 1- initialization expression
 Expression 2- conditional(test) expression
 Expression 3- updation/manipulation expression



```

for (expr1; expr2; expr3)
  statement

```

- A ‘for’ loop is used when a loop is to be executed a known no. of times. We can do the same thing using a ‘while’ loop, but the ‘for’ loop is easier to read and more natural for counting loops.

Ex. : Calculation of sum of 1 to 10 nos.

```

main
{
    int num, sum = 0;
    for (num=1; num<11; num++)
    {
        sum=sum +num;
    }
    printf("sun is = %d", sum);
}

```

So according to this example-

```
for (num =1; num <11; num ++)
```

Here variable num is working as a computer variable.

- num = 1 => when the for statement is executed for the first time, the value of num is to an initial value- 1 so it is an initialization expression.

- o num < 11 => Now the condition (if num is (limit-test expression) less than 11) is tested. Since num is =1, this condition is satisfied & the body of loop ({sum=sum+num}) is executed for the first-time.
 - o After executing the body of loop, num ++ (or num = num +1(updation expression)) is executed & the value of num is increased by 1. (so num = 1+1=2)
 - o Again the condition (num < 11) is tested now num =2, so it is satisfied. So the body of loop is executed for the second time.
 - o Again (num++) updation expression is executed and num =2+1=3.
 - o Again it is tested (num<11), if it is satisfied, body of the loop is executed.
- : : :
: : :

Similarly these steps are repeated until num =10, when further num is increased → num = 10+1=11, now it does not satisfies the condition (num <11) & the loop ends.

- o Now the statement (printf (“sum=%d”, sum)) is executed in the program.

DIFFERENT WAYS OF IMPLEMENTING FOR LOOP

Form	Comment	Meaning(effect)
for (i=0 ; i < 10 ; i++) Statement1;	Single Statement	{ } not required
for (i=0 ;i <10; i++) { Statement1; Statement2; Statement3;}	Multiple Statements within for	{ } required
for (i=0 ; i < 10;i++) ; printf("Hi")	For Loop with no Body (Carefully Look at the Semicolon)	Loop not effective. Hi will be printed once.
for(i=0,j=0;i<100,j<30;i++,j++) Statement1;	Multiple initialization & Multiple Update Statements Separated by Comma	Comma operator will work from L to R
for (; i<10 ; i++)	Initialization not used	i is initialized before loop
for (; i<10 ;)	Initialization & Update not used	i is initialized before loop and updated after loop
for (; ;)	Infinite Loop, It Never Terminates	

base-10 *n*-narcissistic/Armstrong numbers

1-digit Armstrong numbers	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3-digit Armstrong numbers	153, 370, 371, 407
4-digit Armstrong numbers	1634, 8208, 9474
5-digit Armstrong numbers	54748, 92727, 93084

6-digit Armstrong numbers	548834
7-digit Armstrong numbers	1741725, 4210818, 9800817, 9926315
8-digit Armstrong numbers	24678050, 24678051, 88593477
9-digit Armstrong numbers	146511208, 472335975, 534494836, 912985153
10-digit Armstrong numbers	4679307774

Nesting of loop: Here one *for* loop is used within another *for* loop as:

```
main()
{
int r, c, sum;
for(r=1; r<4;r++)
{
    for(c=1;c<3;c++)   ← outer loop
    {
        sum=r+c;
        printf("r=%d c=%d sum=%d", r,c,sum);
    }
}
}
```

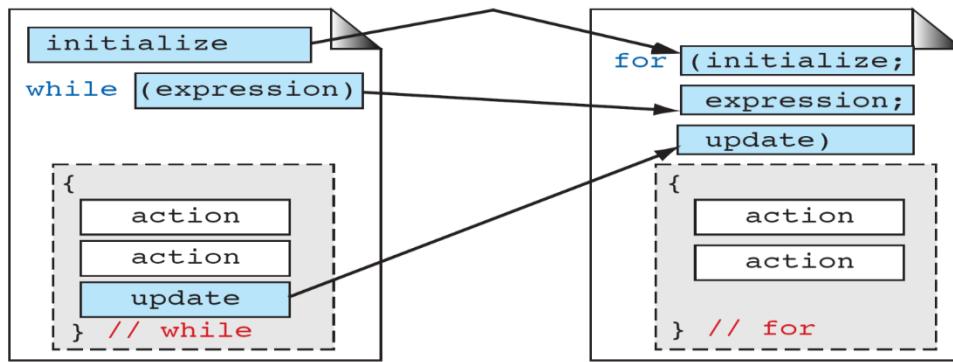
o/p of above program:-

r = 1	c = 1	sum = 2
r = 1	c = 2	sum = 3
r = 2	c = 1	sum = 3
r = 2	c = 2	sum = 4
r = 3	c = 1	sum = 4
r = 3	c = 2	sum = 5

- So here we can see for each value of r (outer loop) all the values of (inner loop) are executed.
- Value of r is incremented only if for its first value, all values of c are being executed & inner loop is terminated.

Comparing while & for loops.

A *for* loop is used when a loop is to be executed a known number of times. We can do the same with a *while* loop, but the *for* loop is easier to read and more natural for counting loops.



while

```
main()
{
    int i=1, int a;
    int sum=0;
    while (i<=20)
    {
        scanf("%d", &a);
        sum +=a;
        i++;
    }
    printf("sum is=%d", sum);
}
```

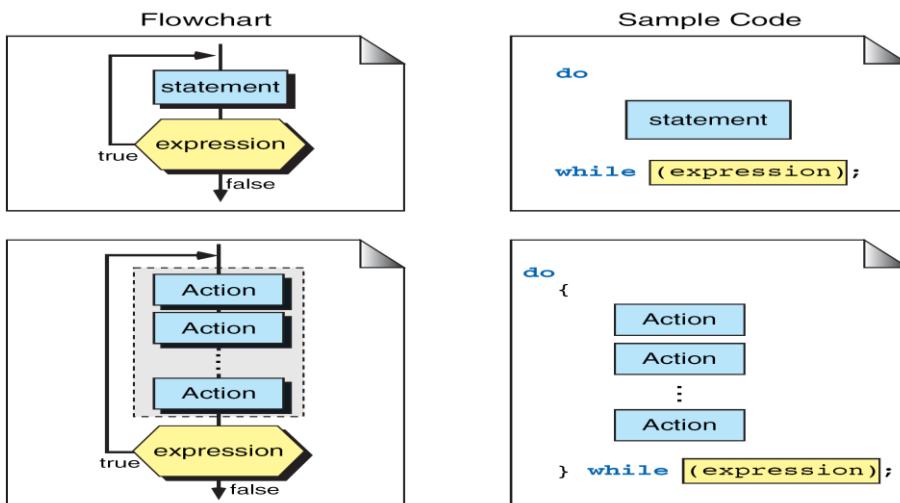
for

```
main()
{
    int a, sum = 0;
    int i;
    for (i=1; i<=20; i++)
    {
        scanf("%d", &a);
        sum +=a;
    }
    printf("sum is=%d", sum);
```

- Above program is: Read 20 nos. from keyboard and find their sum.

The do.....while loop: It is a post-test (exit controlled) loop:

Flow chart-



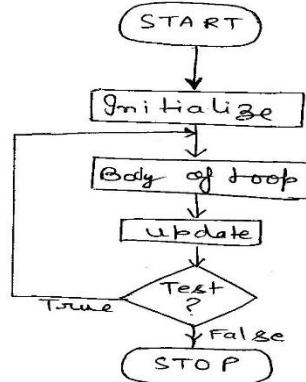
Syntax:

```
do
{
    Body of the loop
} while (test_cond.);
```

It is used generally when we do not know **already** that how many times the loop will be executed (repeated).

Do...while loop is concluded with a semicolon ‘;’.

It is a **post-test loop** so that the body of the loop is always executed **at least once**, then only the expression/test-condition is evaluated, so either test_cond → true **or** false body of loop is already executed once.



eg. Sum of no. 1 to 10 using do-while :-

```
main()
{
    int num=1, sum=0;
    do
    {
        sum=sum+num;
        num++;
    }while (num<11);
    printf("sum is=%d", sum);
}
```

So in above program loop will execute at least once even if we initialize num=11.

Example:

Write a C program to add all the numbers entered by a user until user enters 0.

```
main()
{
    int sum=0,num;
    do      /* Codes inside the body of do...while loops are at least executed once. */
    {
        printf("Enter a number\n");
        scanf("%d",&num);
        sum+=num;
    } while(num!=0);
    printf("sum=%d",sum);
}
```

Program.: enter a no. & square it until a user says → No

Using – do....while loop

```
main()
{
char next;
int num;

do{
    printf("Enter a number");
    scanf("%d", & num);
    printf("square of %d is %d", num, num *
num);
fflush(stdin);//flush buffer to remove:
    \n
    printf ("\\nIf u want to square another
number -press 'y' otherwise press any char
");
    scanf("%c", &next);
}while(next == 'y');
}
```

Using for loop

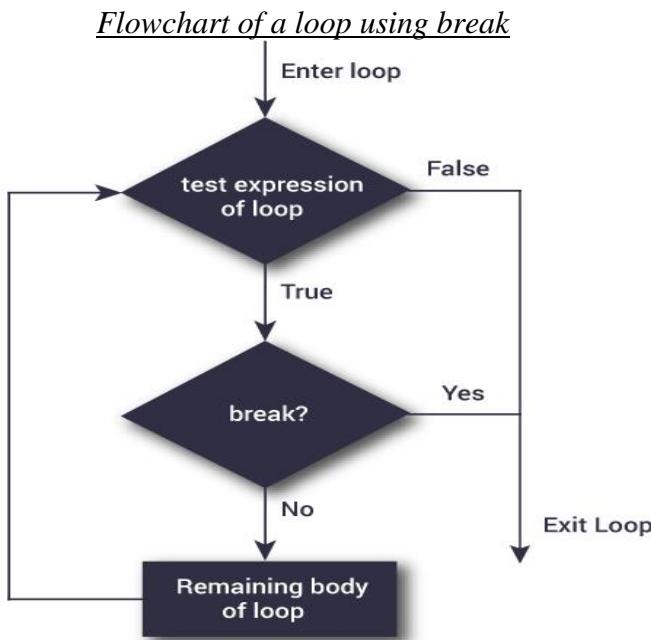
```
main()
{
char next;
int num;

for(next = 'y';next== 'y'; scanf("%c",
&next))
{
    printf("Enter a no.");
scanf("%d%*c", &num); //flush
buffer
    printf("square of %d is %d", num,
num*num);
    printf("\\nIf u want to enter next number
press-'y' otherwise press any char:");
}}
```

Break statement:-

An early exit from a loop can be performed using the break statement.

- In a loop, the break statement causes a loop to terminate. It is similar to condition (test-expression) of loop results-*false*.
- If we are in a series of nested loops, ‘break’ terminates only the inner loop-*the one we are currently in*.
- Break statement can be used in any of the loop statements-while, for, and do....while- and in the selection switch statement.
- Generally break is associated with an ‘ if ’.



USE OF BREAK:

```
main()
{
    int next = 1; int num;
    for(;next== 1;)
    {
        printf("Enter a no.");
        scanf("%d",&num);
        if(num)
            printf("square of %d is %d", num, num*num);
        else break; } }
```

eg.: WAP to test a no. if it is prime or not.

- A no. is prime if it is divisible by only 1 & itself otherwise it is not divisible by any no. so

Solution 1:without using flag variable

```
main()
{
```

Solution 2: using flag variable

```
main()
{
```

<pre> int num, i; printf("Enter no."); scanf("%d", &num); i=2; while(i <=num-1) { if (num%i==0) { printf("Not a prime number"); break;} i++; } if(i==num) printf("Prime number"); } </pre>	<pre> int n, i, flag=0; printf("Enter a positive integer: "); scanf("%d",&n); for(i=2;i<=n-1;i++) { if(n%i==0) { flag=1; break; } } if (flag==0) printf("%d is a prime number.",n); else printf("%d is not a prime number.",n); } </pre>
---	---

Here **break**, takes the control out only from the ‘while’ in which it is placed loops iterations will be stopped.

“Continue” statement. (*Skipping a part of a loop*):

“Continue” tells the compiler – “Skip the following statement within loop and continue with the next iteration.”

When the ‘continue’ is encountered inside any loop, control automatically passes to the beginning of the loop.

eg:

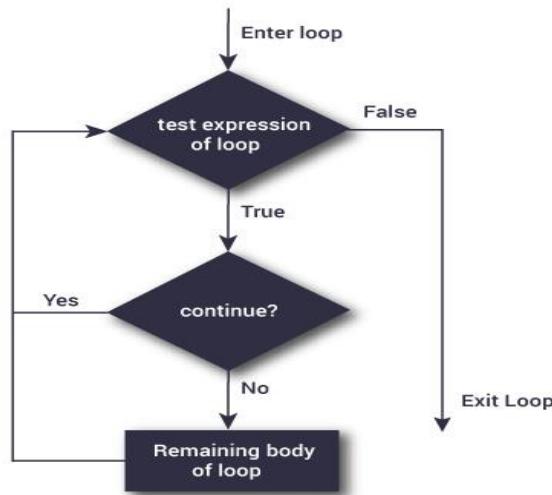
```

main()
{
    int i, j;
    for(i=1;i<=2;i++)
    {
        for(j=1;j<=2;j++)
        {
            if(i == j)
                continue;
            printf("\n%d%d\n",i,j); }
        }
    }

```

o/p → 1 2
 2 1

Flowchart of a loop using continue



*WAP to print n odd numbers using continue statement

Note:- When value of i=value of j, the `continue` statement takes the control to the for loop(inner) by passing rest of the statement pending execution in the inner for loop – `printf` statement.

Difference between Break & Continue

Break	Continue
When <code>break</code> is executed, the statement following <code>break</code> are skipped and cause the loop to be terminated.	When <code>continue</code> statement is executed, the statements following <code>continue</code> are skipped and cause the loop to be continued with the next iteration.
It can be used in switch statement to transfer the control outside switch.	It cannot be used inside a switch statement. It is used only with loops.
For example, <code>for(i=1;i<=5;i++)</code> <code>{</code> <code> if(i%2==0) break;</code> <code> printf("%d",i);</code> <code>}</code>	For example, <code>for(i=1;i<=5;i++)</code> <code>{</code> <code> if(i%2==0) continue;</code> <code> printf("%d",i);</code> <code>}</code>
Output 1	Output 135

Write a C Program to print half pyramid as using * as shown in figure below.

<pre> * * * * * * * * * * * * * * * rows=5 for above output </pre>	<pre> main() { int i,j,rows; printf("Enter the number of rows: "); scanf("%d",&rows); for(i=1;i<=rows;++i) { for(j=1;j<=i;++j) { printf("* "); } printf("\n"); } } </pre>
--	---

Write a C Program to print half pyramid as using numbers as shown in figure below.

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

```
main()  
{  int i,j,rows;  
    printf("Enter the number of rows: ");  
    scanf("%d",&rows);  
    for(i=1;i<=rows;++i)  
    {      for(j=1;j<=i;++j)  
        {          printf("%d ",j);      }  
        printf("\n");    }}
```

Write a C Program to print triangle of characters as below

```
A  
B B  
C C C  
D D D D  
E E E E E
```

```
main()  
{  
    int i,j;  
    char ch,temp='A';  
    printf("Enter uppercase character you want in triangle at  
last row: ");  
    scanf("%c",&ch);  
    for(i=1;i<=(ch-'A'+1);++i)  
    {      for(j=1;j<=i;++j)  
        printf("%c",temp);  
        ++temp;  
        printf("\n");  
    } }
```

Write a C Program to print inverted half pyramid using * as shown below.

```
* * * * *  
* * * *  
* * *  
* *  
*
```

```
main()  
{  
    int i,j,rows;  
    printf("Enter the number of rows: ");  
    scanf("%d",&rows);  
    for(i=rows;i>=1;--i)  
    {  
        for(j=1;j<=i;++j)  
        {  
            printf("*");        }  
        printf("\n");  
    } }
```

Write a C program to print pyramid using *.

```
      *  
     * * *  
    * * * * *  
   * * * * * * *  
* * * * * * * * *
```

```
main()  
{  
    int i,j,space,rows,k=1;  
    printf("Enter the number of rows: ");  
    scanf("%d",&rows);  
    for(i=1;i<=rows;++i)  
    {  
        for(j=1;j<=rows-i;++j)  
        { printf(" ");}  
        for(j=1;j<=2*i-1;++j){printf("*");}        }  
        printf("\n");  
    }     printf("\n");  
}
```

Write a C program to print the pyramid of digits in pattern as below.

```
1  
2 3 2
```

```
main()  
{  
    int i,space,rows,k=0,count=0,count1=0;  
    printf("Enter the number of rows: ");
```

```

3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5

```

```

scanf("%d",&rows);
for(i=1;i<=rows;++i)
{ for(space=1;space<=rows-i;++space)
  { printf(" "); ++count; }
  while(k!=2*i-1)
  { if (count<=rows-1)
    { printf("%d ",(i+k));
      ++count; }
    else
    { ++count1;
      printf("%d ", (i+k-2*count1)); }
    ++k; }
  count1=count=k=0; printf("\n"); }
}

```

WRITE A C PROGRAM TO DISPLAY REVERSE PYRAMID.

```

* * * * * * *
* * * * * *
* * * * *
* * *
*

```

```

main()
{
int rows,i,j,space;
printf("Enter number of rows: ");
scanf("%d",&rows);
for(i=rows;i>=1;--i)
{
  for(space=0;space<rows-i;++space)
    printf(" ");
  for(j=i;j<=2*i-1;++j)
    printf("* ");
  for(j=0;j<i-1;++j)
    printf("* ");
  printf("\n"); }
}

```

C PROGRAM TO DRAW PASCAL'S TRIANGLE (USING FUNCTONS)

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

```

//logic to get factorial of a number
long fact(int num){
  long f=1;
  int i=1;
  while(i<=num){
    f=f*i;
    i++; }
  return f;
}
int main(){
int row,i,j,k;printf("Enter the no. of lines: ");
scanf("%d",&row);
for(i=0;i<row;i++)
{ for(j=0;j<row-i-1;j++)
  { printf(" ");}
  for(k=0;k<=i;k++)
//formula to get binomial
coefficient(iCk):fact(i)/(fact(k)*fact(i-k))
  { printf("%2d ",fact(i)/(fact(k)*fact(i-k))); }
  printf("\n");
} }

```

C PROGRAM TO DISPLAY FLOYD'S TRIANGLE.

```

1
2 3
4 5 6
7 8 9 10

```

```

main()
{
    int rows,i,j,k=1;
    printf("Enter number of rows: ");
    scanf("%d",&rows);
    for(i=1;i<=rows;i++)
    {
        for(j=1;j<=i;j++)
        {
            printf("%4d",k);
            ++k;
        }
        printf("\n");
    }
}

```

Definition: perfect number

Perfect number is a positive number which is sum of all positive divisors excluding that number is equal to that number.

For example 6 is perfect number since divisor of 6 are 1, 2 and 3. Sum of its divisor is

$$1 + 2 + 3 = 6$$

Note: 6 is the smallest perfect number.

Next perfect number is 28 since $1 + 2 + 4 + 7 + 14 = 28$. Other perfect numbers: **496, 8128**

Definition of strong number:

A number is called strong number if sum of the factorial of its digits is equal to number itself. For example: 145 since $1! + 4! + 5! = 1 + 24 + 120 = 145$, 40585 is also a strong number.

DEFINITION :ARMSTRONG NUMBER:

Armstrong number is a number which is equal to sum of digits raise to the power total number of digits in the number. Some Armstrong numbers are: 0, 1, 2, 3, 153, 370, 407, 1634, 8208 etc.

Example 1: 153: Total digits in 153 is 3, And $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

Sum of $1^2 + 2^2 + \dots + n^2$ series in c program

```

#include<stdio.h>
int main(){
    int n,i;
    int sum=0;
    printf("Enter the n i.e. max values of series: ");
    scanf("%d",&n);
    sum = (n * (n + 1) * (2 * n + 1)) / 6;
    printf("Sum of the series : ");
    for(i=1;i<=n;i++){
        if (i != n)
            printf("%d^2 + ",i);
        else
            printf("%d^2 = %d ",i,sum);    }
}

```