

UNIT-II

CONTROL FLOW STATEMENTS

Index

2.1. Types of control flow statements

2.2. Branching / selection / decision control statements

- if
- if-else
- if-else-if
- nested if
- switch

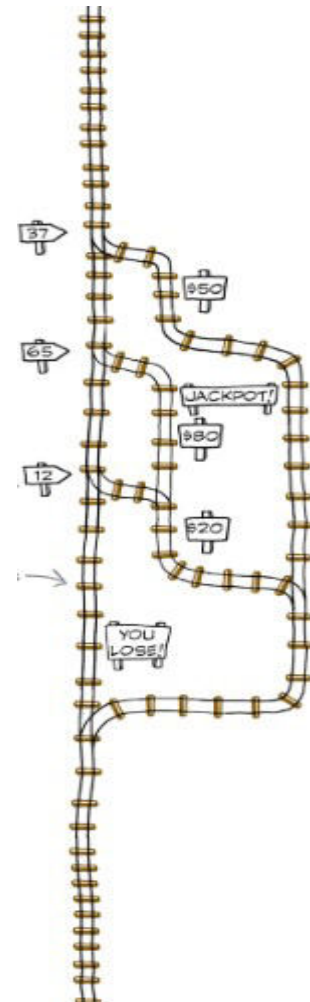
2.3. Looping / repetitive statements

- while
- do-while
- for

2.4. Jumping / transfer statements

- break
- continue
- goto

2.5. Nested loops with examples



CONTROL FLOW STATEMENTS

Flow control:

Flow control describes the order in which all the statements will be executed at run time.

C control Statements:-

There are three types of flow control statements in C

1. Selection Statements / branching statements / decision control statements
2. Iteration statements/Looping statements
3. Transfer statements/ Jumping statements

Diagram:

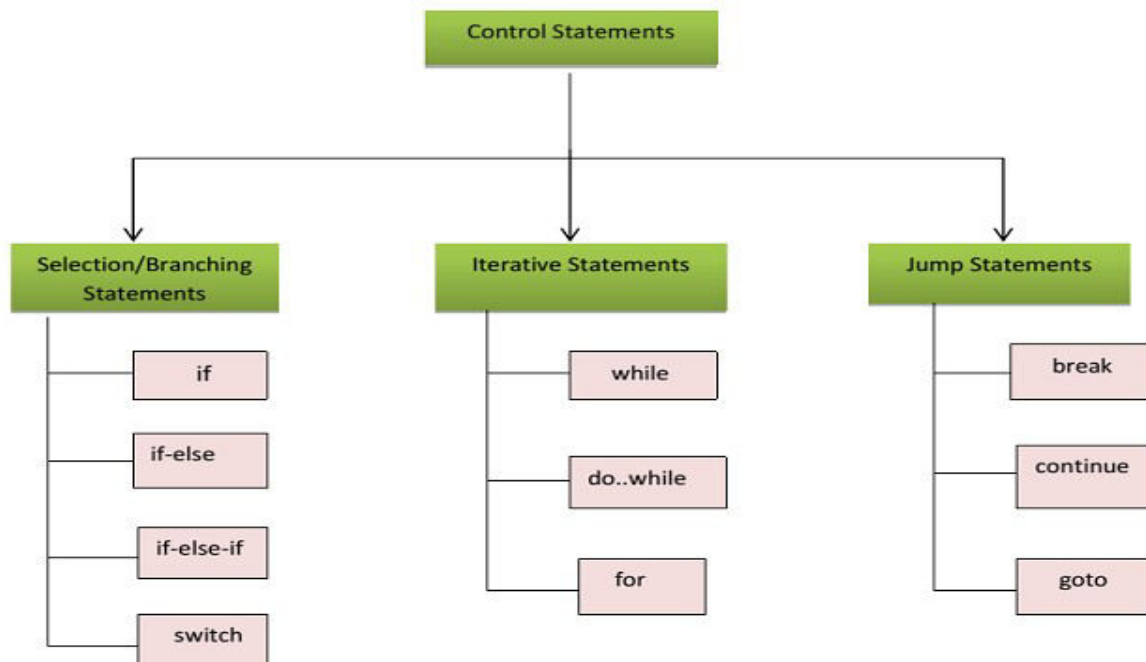


Figure: Types of control flow statements

DECISION CONTROL STATEMENTS

(Selection statements or Branching Statements)

The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not. These decision control statements include:

- I) if statement
- II) switch statement

I) if statement:

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are

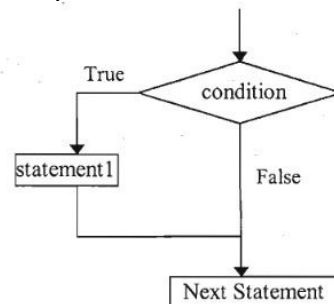
1. simple if statement
2. if-else statement
3. Nested if statement)
4. else if ladder statement

1. simple if statement:

The if statement is the simplest form of decision control statements that is frequently used in **decision making**. The general form of a simple if statement is shown below.

Syntax of if statement

```
if(test expression)
{
    statement 1;
    .....
    statement n;
}
statement x;
```



Flow chart of if control statement

The **if** structure may include one statement or **n** statements enclosed within braces. First the test expression is evaluated. If the test expression is true, the statement of if block(statements 1 to n) are executed, otherwise these statements will be skipped and the execution will jump to statement x.

Note that there is no semicolon after the test expression. This is because the condition and statement should be placed together as a single statement.

Example: program to determine whether a person is eligible to vote.

```
#include<stdio.h>
void main()
{
    int age;
    printf("\n Enter the age:");
    scanf("%d",&age);
    if(age>=18)
        printf("\n You are eligible to vote");
}
```

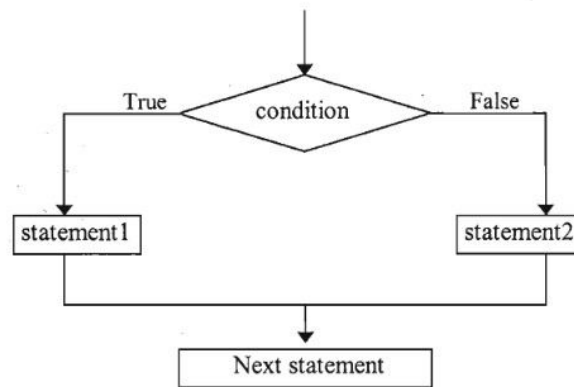
Output:

Enter the age:28
You are eligible to vote

2. if-else statement: The if...else statements is an extension of the simple if statement. The general form is

```
if( test expression)
{
    True-block statement(s)
}
else
{
    False-block statement(s)
}
statement-x
```

If the *test expression* is true, then the true-block *statement(s)*, immediately following the **if** statements are executed; otherwise, the *false-block statement(s)* are executed.



Flow chart of if...else control statement

In both the cases, the control is transferred subsequently to the statement-x.

Example: Write a program to find whether the given number is even or odd.

```
#include<stdio.h>
void main()
{
    int num;
    printf("\n Enter any number:");
    scanf("%d",&num);
    if(num%2==0)
        printf("\n %d is an even number",num);
    else
        printf("\n%d is an odd number",num);
}
```

OUTPUT:

Enter any number:11
11 is an odd number

3. nested if statement:

if statement inside an if statement is known as nested if. if statement in this case is the target of another if or else statement. When more then one condition needs to be true and one of the

condition is the sub-condition of parent condition, **nested if** can be used. This takes the following general form:

```
if(test condition-1)
{
    if(test condition-2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-x;
```

The logic of execution is: If the *condition-1* is false, the statement-3 will be executed; otherwise it continues to perform the second test. If the *condition-2* is true, the *statement-1* will be evaluated ; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

Example: The program selects and prints the largest of the three numbers using nested if....else statements

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter three values\n");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
            printf("%d\n",a);
        else
            printf("%d\n",c);
    }
    else
    {
        if(c>b)
            printf("%d\n",c);
        else
            printf("%d\n",b);
    }
}
```

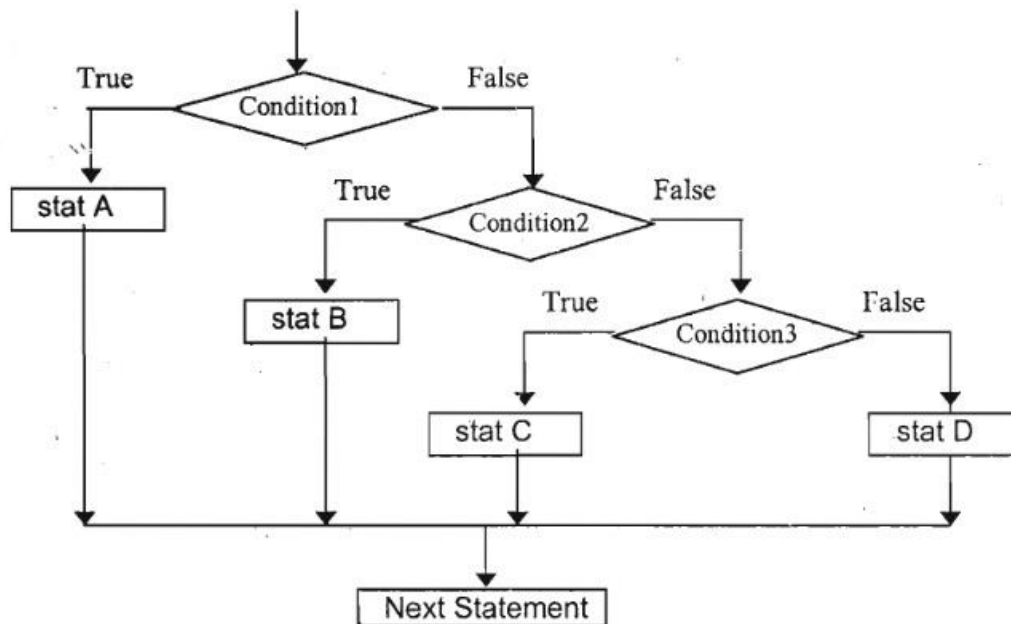
Output:

```
Enter three values
10    20    15
Largest value is 20
```

4. else if ladder statement:

There is another way of putting **ifs** together when multipath decisions are involved. A multipath decision is a chain of **ifs** in which the statement associated with each **else** is an **if**. It takes the following general form.

```
if(condition 1)
    statement-1;
else if(condition 2)
    statement-2;
else if(condition 3)
    statement-3;
else if(condition n)
    statement-n;
else
    default statement;
statement-x;
```



Flow chart of else...if ladder

This construct is known as the **else if** ladder. The conditions are evaluated from the top(of the ladder), downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x(skipping the rest of the ladder). When all the n-conditions become false, then the final **else** containing the *default-statement* will be executed. Figure shows the logic of execution of **else if** ladder statements.

Example: Program onto display the examination result.

```
#include<stdio.h>
void main()
{
    int marks;
    printf("\nEnter the marks obtained:");
    scanf("%d",&marks);
    if(marks>=75)
        printf("\n DISTINCTION");
    else if(marks>=60 && marks<75)
        printf("\n FIRST DIVISION");
    else if(marks>=50 && marks<60)
        printf("\n SECOND DIVISION");
    else if(marks>=40 && marks<50)
        printf("\n THIRD DIVISION");
    else
        printf("\n FAIL");
}
```

Output:

```
Enter the marks obtained:70
FIRST DIVISION
```

II. switch statement:

C has a built in multi way decision statement known as a switch. The switch statement tests the value of a given variable(or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed. The general form of the switch statement is as shown below

Syntax of switch statement:

```
switch(variable)
{
    case value1:
        Statement block1;
        break;
    case value2:
        Statement block2;
        break;
    .....
    case valueN:
        Statement blockN;
        break;
    default:
        default statement;
        break;
}
Statement X;
```

Here **switch**, **case** and **default** are keywords. The "expression" following the switch keyword can be any C expression that yields an integer value. It can be value of any **integer** or **character** variable, or a function call returning an integer, or an arithmetic, logical, relational,

bitwise expression yielding an integer. It can be any **integer** or **character constant** also. Since characters are converted to their ASCII values, so we can also use characters in this expression. Data types **long int** and **short int** are also allowed. The constants following the case keywords should be of **integer** or **character** type. They can be either constants or constant expressions. These constants must be different from one another.

We can't use floating point or string constants. Multiple constants in a single **case** are not allowed; each **case** should be followed by only one constant.

Each **case** can be followed by any number of statements. It is also possible that a **case** has no statement under it. If a **case** is followed by multiple statements, then it is not necessary to enclose them within pair of curly braces, but it is not an error if we do so. The statements under case can be any valid C statements like **if else**, **while**, **for** or even another **switch** statement. Writing a **switch** statement inside another is called nesting of **switches**. some valid and invalid ways of writing **switch** expressions and **case** constants.

int a, b, c; char d, e; float f;

Valid

switch(a) switch(a>b) switch(d +e-3) switch(a>b && b>c) switch(func(a, b))

Invalid

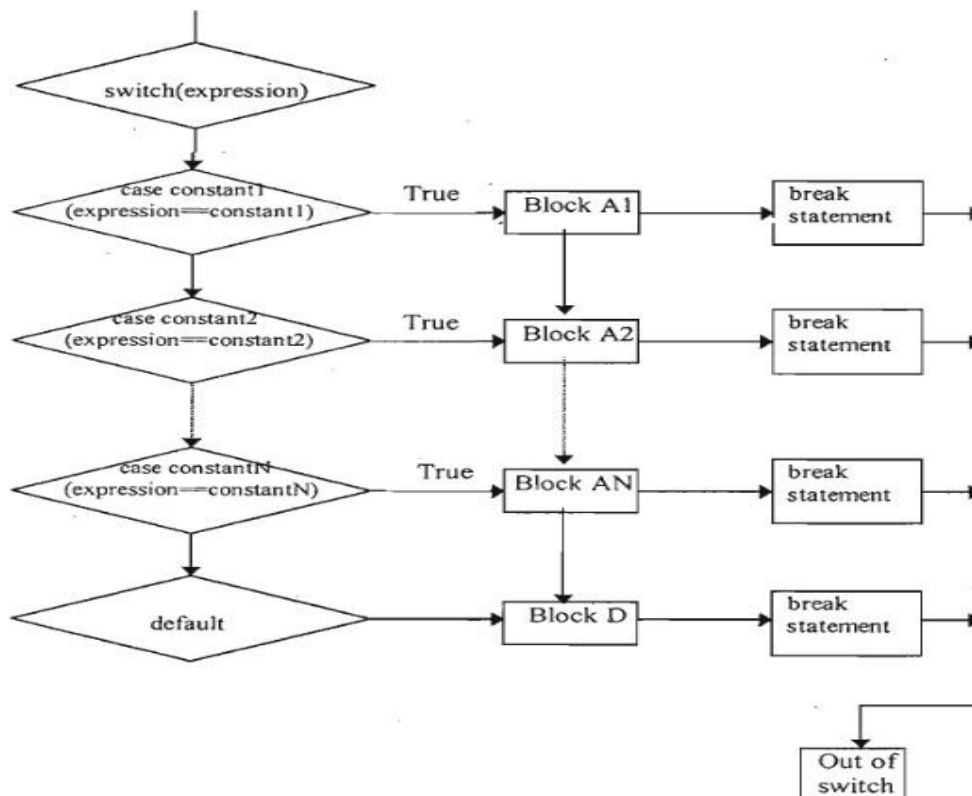
switch(f) switch(a+4.5)

Valid

case 4: case 'a': case 2+4: case 'a'>'b':

Invalid

case "second": case 2.3: case a: case a>b: case a+2: case 2, 4, 5:
case 2 : 4 : 5 :



Example: program to determine whether entered character is vowel or not.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("\n Enter any character:");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'a':
            printf("\n%c is a VOWEL",ch);
            break;
        case 'e':
            printf("\n%c is a VOWEL",ch);
            break;
        case 'i':
            printf("\n%c is a VOWEL",ch);
            break;
        case 'o':
            printf("\n%c is a VOWEL",ch);
            break;
        case 'u':
            printf("\n%c is a VOWEL",ch);
            break;
        default:
            printf("\n%c is not a VOWEL",ch);
            break;
    }
}
```

OUTPUT:

Enter any character:e
e is a VOWEL

LOOPING STATEMENTS

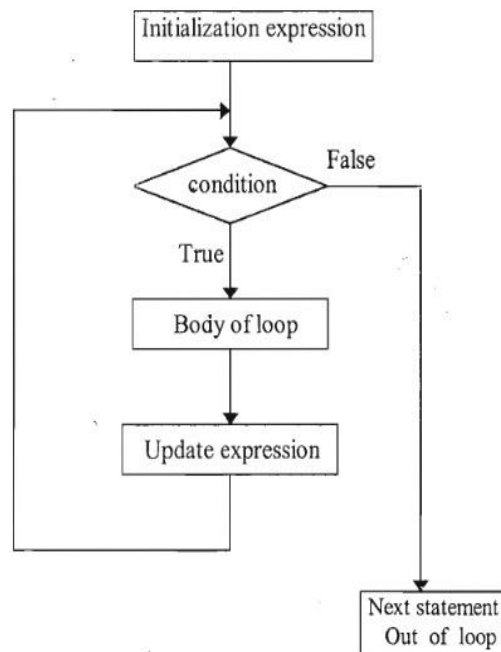
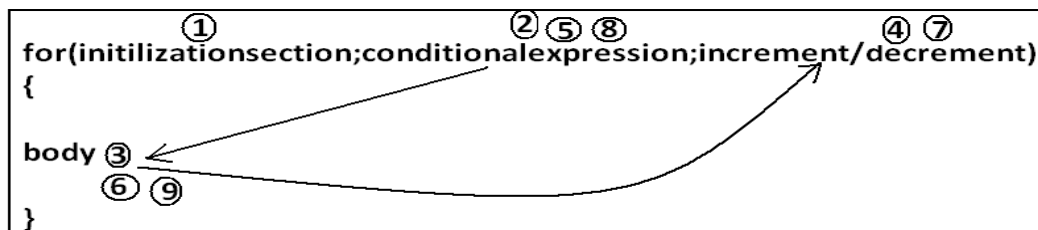
When a single statement or group of statements will be executed again and again in a program then such type processing is called Loop(or iteration). The C programming language contains 3 different programming statements for program looping.

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

1. for loop:

It is a looping statement which repeat again and again till it satisfies the condition it is also entry control loop.

Syntax:



Flow chart of for loop

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test condition is false, for loop is terminated. But if test condition is true then the code/s inside body of for loop is executed and then update expression is updated. In the above syntax, update expression is increment/decrement. This process repeats until test condition is false.

Example: Program to display from 1 to 10 using for loop

```
#include<stdio.h>
void main( )
{
    int i;
    for(i=1;i<=10;i++)
    {
        printf("%d\t",i);
    }
}
```

Output:

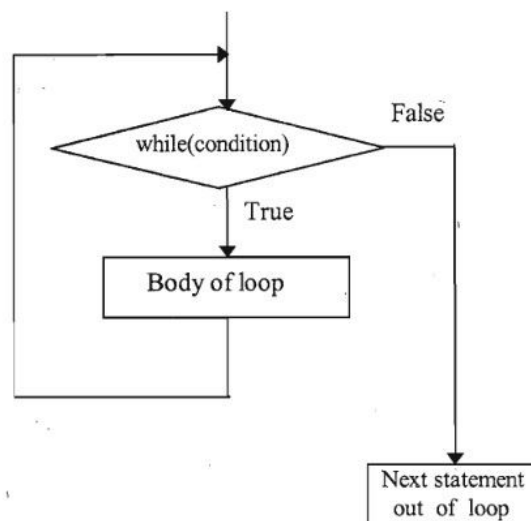
1 2 3 4 5 6 7 8 9 10

2. while loop:

The while loop is best suited to repeat a statement or a set of statement as long as some condition is satisfied. The general form of while loop is

```
while(test condition)
{
    Body of the loop
}
```

The while is an entry controlled loop statement. The test condition is evaluated and if the condition is true, then body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process continues until the test-condition finally becomes false the control is transferred out of the loop and the body of the loop may have one or more statements. If the test condition evaluate to false at the first time the statements are never executed.



Flow chart of while loop

Example: Program to calculate factorial of a given number using while loop.

```
#include<stdio.h>
void main( )
{
    int num,fact=1;
    printf("Enter any number");
    scanf("%d",&num);
    while(num>0)
    {
        fact=fact*num;
        num--;
    }
    printf("Factorial of the given number=%d",fact);
}
```

Output:

Enter any number=5
Factorial of the given number=120

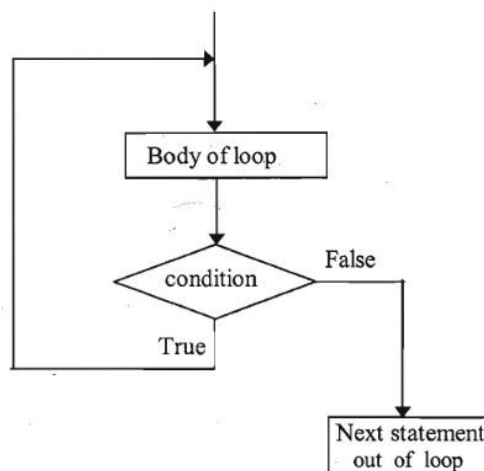
3. do-while Loop:

The while loop makes a test of condition before the loop is executed. Therefore the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. On some occasions, it might be necessary to execute the body of the loop before the test is performed. such situations can be handle with the help of the body of the do-while statement.

Syntax:

```
do
{
    body of the loop
}while(test condition); //semicolon is mandatory
```

In this statement first body of the loop is executed and then the condition is checked. If condition is true then body of the loop is executed when condition becomes false then it will exit from the loop. Note that semicolon must be at the end of the loop.



Flow chart of do ... while loop

Example: Write a program to calculate the average of first n numbers

```
#include<stdio.h>
void main()
{
    int n,i=0,sum=0;
    float avg=0.0;
    printf("\nEnter the value of n:");
    scanf("%d",&n);
    do
    {
        sum=sum+i;
        i=i+1;
    }while(i<=n);
    avg=sum/n;
    printf("\n The sum of the first n numbers=%d",sum);
    printf("\n The average of first %d numbers=%f",n,avg);
}
```

Output:

Enter the value of n:18
The sum of first n numbers=171
The average of first %d numbers=9.00

TRANSFER STATEMENTS

By using transfer statements, we are able to transfer the flow of execution from one position to another position.

1. break
2. continue
3. goto

1) **break:**

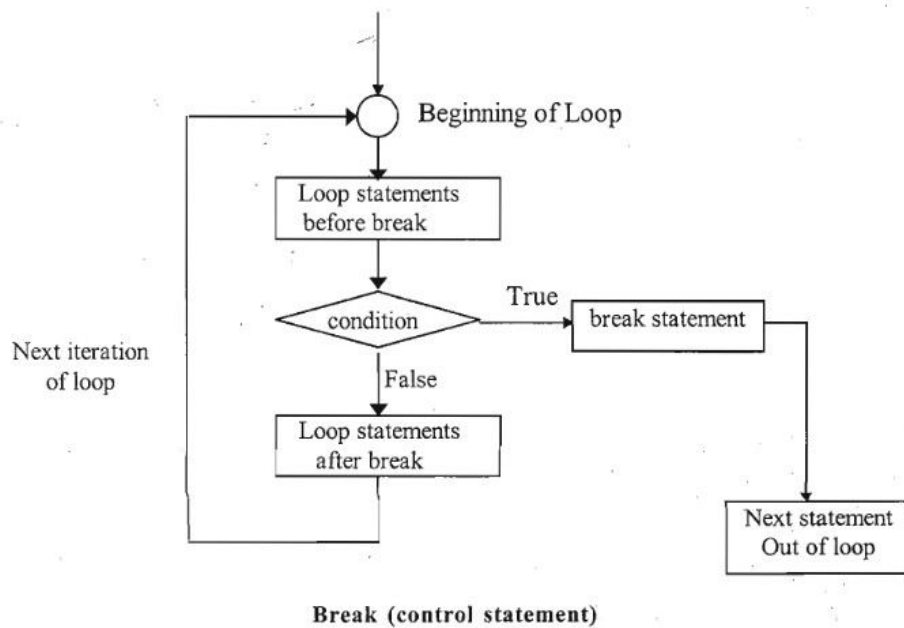
In C programming, break is used to stop the execution. We are able to use the break statement in only two places.

- a) **Inside the switch statement**
- b) **inside the loops**

If we are using any other place, the compiler will generate compilation error message “**break outside switch or loop**”.

The general syntax for break statement is

Syntax: **break;**



Example of break statement:

```
#include<stdio.h>
void main()
{
    int i=0;
    while(i<=10)
    {
        if(i==5)
            break;
        printf("%d\t",i);
        i=i+1;
    }
}
```

Output:

1 2 3 4

Example: If we are using break outside switch or loops the compiler will raise compilation **error** “break outside switch or loop”.

```
#include<stdio.h>
void main()
{
    if(1)
    {
        printf("krishna");
        break;
        printf("karthik");
    }
}
```

Output:

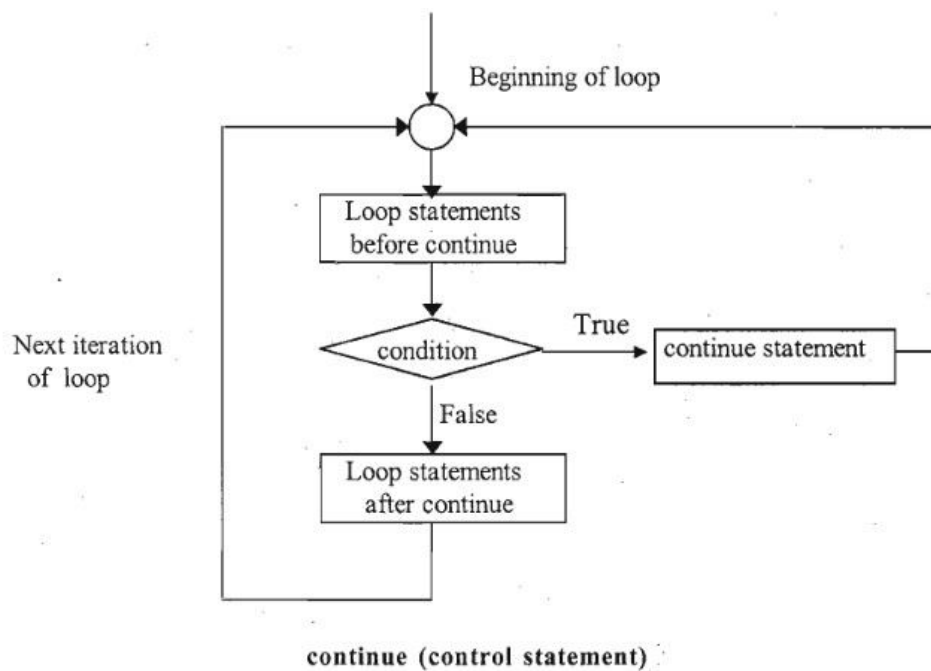
error: break statement not within loop or switch

2) continue:

It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used. The general syntax is

syntax:

```
continue;
```



Example of continue statement:

```

#include<stdio.h>
void main()
{
    int i;
    for(i=1;i<=10;i++)
    {
        if(i==5)
            continue;
        printf("%d\t",i);
    }
}

```

Output:

1 2 3 4 6 7 8 9 10

3) goto :

This is an unconditional control statement that transfers the flow of control to another part of the program.

The goto statement can be used as-

```

goto label;

.....

label:
    statement;

.....

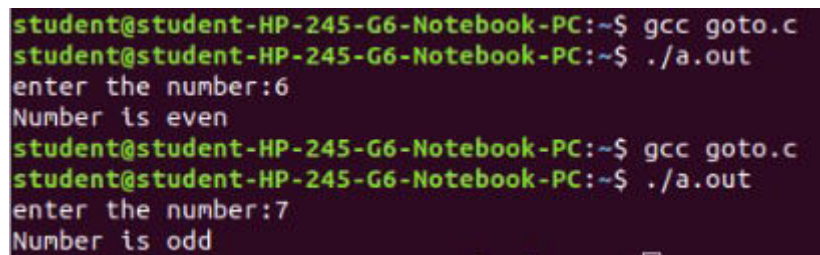
```


Here **label** is any valid C identifier and it is followed by a colon.

Whenever the statement **goto label;** is encountered, the control is transferred to the statement that is immediately after the label.

```
/*Program to print whether the number is even or odd*/
#include<stdio.h>
void main()
{
    int n;
    printf("enter the number:");
    scanf("%d",&n);
    if(n%2==0)
        goto even;
    else
        goto odd;
    even:
        printf ("Number is even");
        goto end;
    odd:
        printf ("Number is odd");
        goto end;
    end:
        printf("\n");
}
```

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ gcc goto.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the number:6
Number is even
student@student-HP-245-G6-Notebook-PC:~$ gcc goto.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter the number:7
Number is odd
```

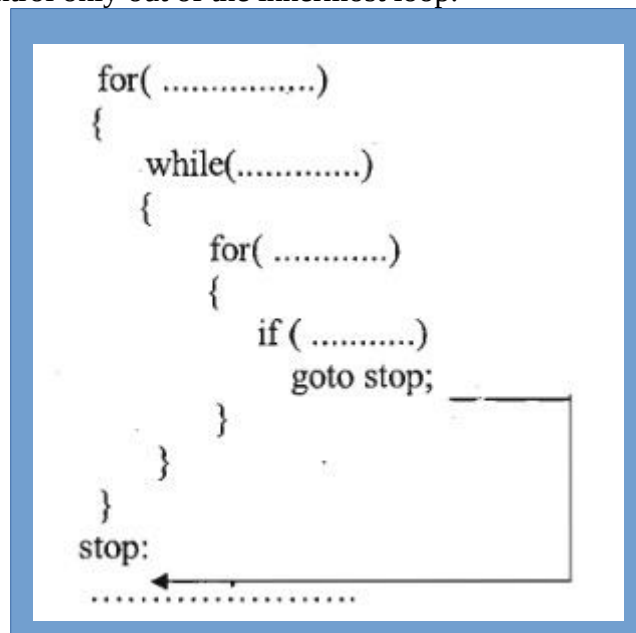
The **label** can be placed anywhere. If the **label** is after the **goto** then the control is transferred forward and it is known as forward jump or forward goto, and if the **label** is before the **goto** then the control is transferred backwards and it is known as backward jump or backward goto. In forward goto, the statements between **goto** and **label** will not be executed and in backward goto statements between **goto** and **label** will be executed repeatedly. More than one goto can be associated with the same **label** but we **cannot** have same **label** at more than one place.

The control can be transferred only within a function using **goto** statement.

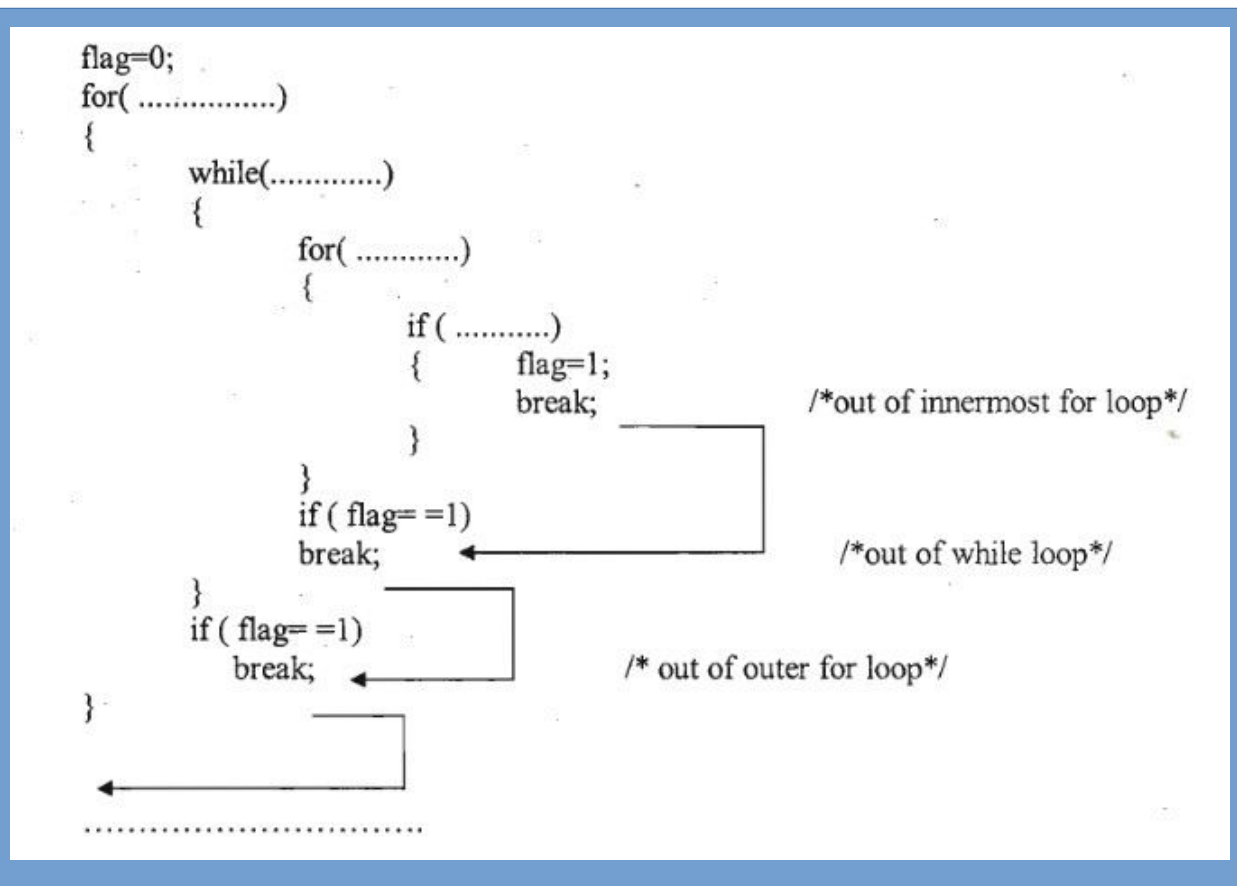
There should always be a statement after any **label**. If **label** is at the end of program, and no statements are to be written after it, we can write the null statement (single semicolon) after the **label** because a program can't end with a **label**.

The use of '**goto**' should be avoided, as it is difficult to understand where the control is being transferred. Sometimes it leads to "spaghetti" code, which is not understandable and is very difficult to debug and maintain. We can always perform all our jobs without using **goto**, and the use of **goto** is not favoured in structured programming.

Although the use of **goto** is not preferred but there is a situation where goto can actually make the code simpler and more readable. This situation arises when we have to exit from deeply nested loops. To exit from a single loop we can use the **break** statement, but in nested loops break will take the control only out of the innermost loop.



we can always write any code without using **goto**, so here also we have another way of exiting out of the deeply nested loop.



We can see that in the case of nested loops, the code using **goto** is more readable and if it is not used, then many tests have to be performed.

NESTED LOOPS

C allows its users to have nested loops, i.e, loops that can be placed inside other loops. Although this feature will work with any loop such as **while**, **do-while** and **for** but it is most commonly used with the **for** loop, because this is easiest to control. A **for** loop can be used to control the number of times that a particular set of statements will be executed. Another outer loop could be used to control the number of times that a whole loop is repeated.

Example:

```
#include<stdio.h>
void main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1;j<=i;j++)
            printf("*");
        printf("\n");
    }
}
```

Output:

\$gcc nested.c

\$/a.out

*

**

Write a C program to print the following pyramid(a)

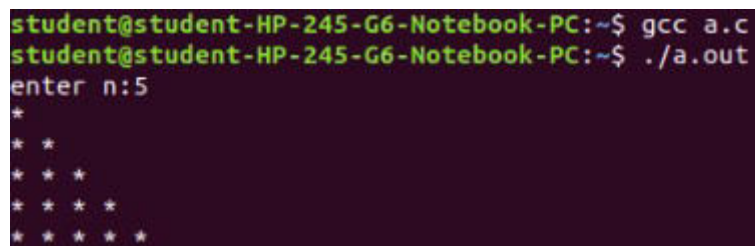
```
*
* *
* * *
* * * *
* * * * *
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)           //loop for number of lines
    {
        for(j=1;j<=i;j++)//loop for printing stars
            printf("* ");
        printf("\n");           //for next line of pyramid
    }
}
```

The outer for loop is for number of lines and the inner loop is for number of stars in each line. We can see that the number of stars is equal to the line number, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Output:



```
student@student-HP-245-G6-Notebook-PC:~$ gcc a.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
*
* *
* * *
* * * *
* * * * *
```

Write a C program to print the following pyramid(b)

```
1
22
333
4444
55555
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)           //loop for number of lines
    {
        for(j=1;j<=i;j++)
            printf("%d",i);
        printf("\n");           //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for number of that row. We can see that the row number is equal to the line number, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here we are printing the value of row number(i)

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc b.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
1
22
333
4444
55555
```

Write a C program to print the following pyramid(c)

```
1
12
123
1234
12345
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)          //loop for number of lines
    {
        for(j=1;j<=i;j++)
            printf("%d",j);
        printf("\n");          //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for number of that column. We can see that the “row number is equal to the line number”, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here we are printing the value of j ie., column number

Output:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)          //loop for number of lines
    {
        for(j=1;j<=i;j++)
            printf("%d",j);
        printf("\n");          //for next line of pyramid
    }
}
```

```
student@student-HP-245-G6-Notebook-PC:~$ gcc c.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
1
12
123
1234
12345
```

Write a C program to print the following pyramid(d)

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n,p=1;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)           //loop for number of lines
    {
        for(j=1;j<=i;j++)
            printf("%3d",p++);
        printf("\n");           //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for printing p value in an increasing order from 1. We can see that the “row number is equal to the line number”, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here we will take a variable p=1 and write the printf statement as printf(“%3d”,p++)

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc d.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```


Write a C program to print the following pyramid(e)

```
2
3 4
4 5 6
5 6 7 8
6 7 8 9 10
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)          //loop for number of lines
    {
        for(j=1;j<=i;j++)// loop for printing sum of row no. and column no.
            printf("%3d",i+j);
        printf("\n");    //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for printing sum of row number(i) and column number(j). We can see that the “row number is equal to the line number”, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here we will print the value of i+j.

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc e.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
 2
3 4
4 5 6
5 6 7 8
6 7 8 9 10
```

Write a C program to print the following pyramid(f)

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)          //loop for number of lines
    {
        for(j=1;j<=i;j++)
        {
            if((i+j)%2==0)
                printf("1 ");
            else
                printf("0 ");
        }
        printf("\n");          //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for printing 1 if sum of row number(i) and column number(j) is even and printing 0 if sum of row number(i) and column number(j) is odd. We can see that the “row number is equal to the line number”, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here we will print 1 if (i+j) is even and print 0 if (i+j) is odd.

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc f.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

Write a C program to print the following pyramid(g)

```
5
54
543
5432
54321
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)          //loop for number of lines
    {
        for(j=1;j<=i;j++)
        {
            printf("%d",n+1-j);
        }
        printf("\n");    //for next line of pyramid
    }
}
```

Here the outer loop is for number of lines and the inner loop is for printing $n+1-j$. We can see that the “row number is equal to the line number”, hence the inner loop will execute once for first line, twice for second line, thrice for third line and so on

Here in this pyramid, we will print $(n+1-j)$.

Second way of the program(g2):

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("enter n:");
    scanf("%d",&n);
    for(i=n;i>=1;i--)          //loop for number of lines
    {
        for(j=n;j>=i;j--)
        {
            printf("%d",j);
        }
        printf("\n");    //for next line of pyramid
    }
}
```

Here we reversed the loops and then printing the value of j

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc g.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
5
54
543
5432
54321
```

```
student@student-HP-245-G6-Notebook-PC:~$ gcc g2.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
enter n:5
5
54
543
5432
54321
```

Write a C program to print the following pyramid(p)

```

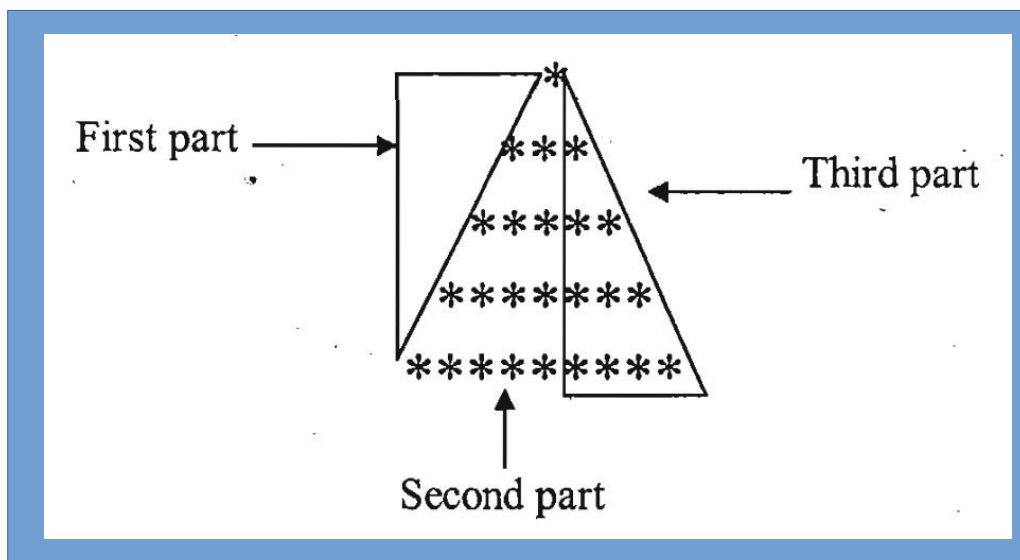
      *
     ***
    *****
   ********
  *********

```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)           //loop for number of lines in pyramid
    {
        for(j=1;j<=n-i;j++)      //loop for spaces(first part)
            printf(" ");
        for(j=1;j<=i;j++)        //loop for second part
            printf("*");
        for(j=1;j<i;j++)         //loop for third part
            printf("*");
        printf("\n");           //for next line of pyramid
    }
}
```



Here first part and second part are written same as in pyramid (n) and third part is written same as in pyramid (a)

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc p.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
  *
 ***
*****
*****
*****
```

Write a C program to print the following pyramid(q)

```

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

```

Program:

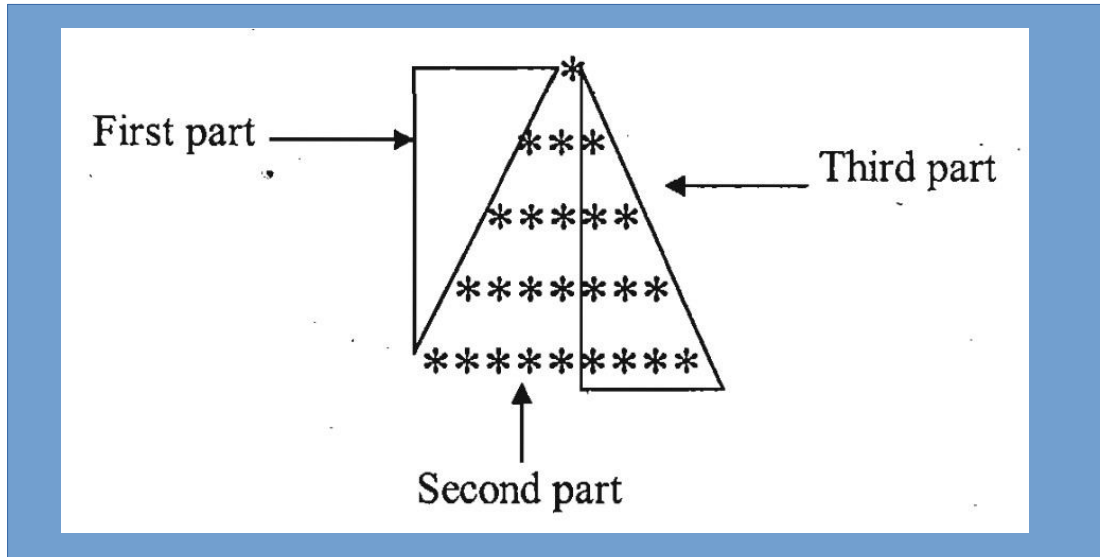
```

#include<stdio.h>
void main()
{
    int i,j,n,p;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)          //loop for number of lines in pyramid
    {
        for(j=1;j<=2*(n+1-i);j++)//loop for spaces(first part)will be from 1 to 2*(n+1-i)
            printf(" ");
        for(j=1;j<=i;j++)      //loop for second part
            printf("* ");
        for(j=1;j<=i;j++)      //loop for third part
            printf("* ");
        printf("\n");          //for next line of pyramid
    }
}

```

The code for pyramid(q) will be the same , only the range of first for loop for spaces will be from 1 to $2*(n+1-i)$ and there will be a space after star in the printf statements.



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc q.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
      *
    * * *
  * * * * *
* * * * * * *
* * * * * * * *
```


Write a C program to print the following pyramid(r)

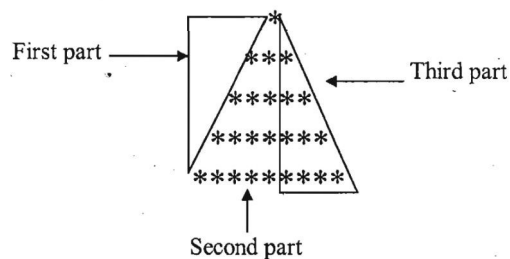
```
  1
 123
12345
1234567
123456789
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n,p;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)                //loop for number of lines in pyramid
    {
        for(j=1;j<=n-i;j++)          //loop for spaces(first part)
            printf(" ");
        p=1;
        for(j=1;j<=i;j++)            //loop for second part
            printf("%d",p++);
        for(j=1;j<=i;j++)            //loop for third part
            printf("%d",p++);
        printf("\n");
    }
}
```

The loop for pyramid (r) will be same as pyramid (p) , but here we will take a variable p and print its value



For pyramid (r) we will initialize the value of p with 1 each time before second inner for loop, and then print the value of p++ in the last two for loops.

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc r.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
  1
 123
12345
1234567
123456789
```

Write a C program to print the following pyramid(s)

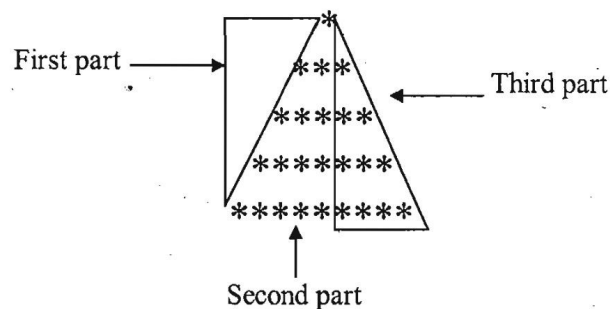
```
1
232
34543
4567654
567898765
```

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n,p;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)           //loop for number of lines in pyramid
    {
        for(j=1;j<=n-i;j++)      //loop for spaces(first part)
            printf(" ");
        p=i;
        for(j=1;j<=i;j++)        //loop for second part
            printf("%d",p++);
        p=p-1;
        for(j=1;j<i;j++)         //loop for third part
            printf("%d",--p);
        printf("\n");
    }
}
```

for pyramid (s) we will initialize the value of p with i before second inner for loop, and then print the value of p++ in second for loop . After this the value of p is decreased by 1 and then the value of --p is printed in the third for loop.



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc s.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
1
232
34543
4567654
567898765
```

Write a C program to print the following pyramid(t)

```

      5
     545
    54345
   5432345
  543212345

```

Program:

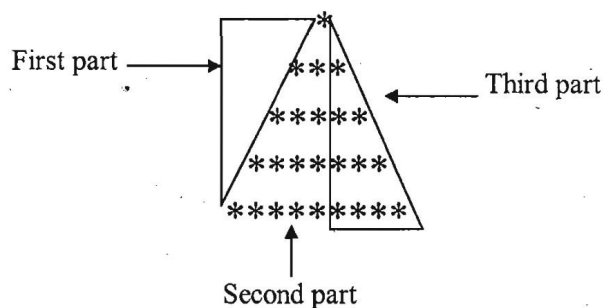
```
#include<stdio.h>
void main()
{
    int i,j,n,p;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)           //loop for number of lines in pyramid
    {
        for(j=1;j<=n-i;j++)     //loop for spaces(first part)
            printf(" ");

        p=n;
        for(j=1;j<=i;j++)       //loop for second part
            printf("%d",p--);

        p=p+2;
        for(j=1;j<=i;j++)       //loop for third part
            printf("%d",p++);
        printf("\n");
    }
}
```

for pyramid (t) we will initialize the value of p with n before second inner for loop, and then print the value of p-- in second for loop and p++ in third loop. The value of p has to be increased by 2 before third for loop.



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc t.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
      5
     545
    54345
   5432345
  543212345

```

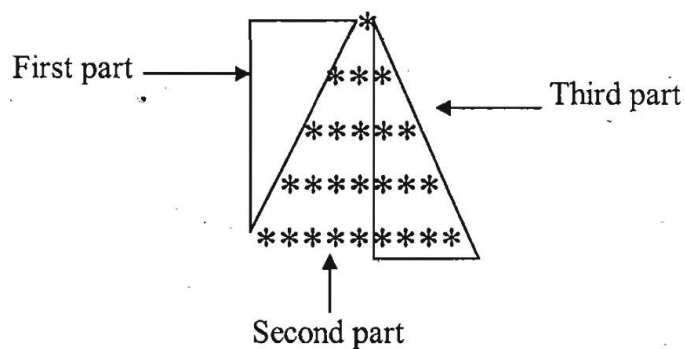
Write a C program to print the following inverted pyramid(u)

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

Program:

```
#include<stdio.h>
void main()
{
    int i,j,n;
    printf("Enter n :");
    scanf("%d",&n);

    for(i=1;i<=n;i++)                //loop for number of lines in pyramid
    {
        for(j=1;j<=i;j++)            //loop for spaces(first part)
            printf(" ");
        for(j=1;j<=(n-i);j++)        //loop for second part
            printf("*");
        for(j=1;j<(n-i);j++)         //loop for third part
            printf("*");
        printf("\n");                //for next line of pyramid
    }
}
```



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc u.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :6
*****
*****
*****
***
*
```

Write a C program to print the following diamond(v)

```

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

```

Diamond (v)

```

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

```

Pyramid (p)

```

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

```

Inverted
pyramid(u)

Description:

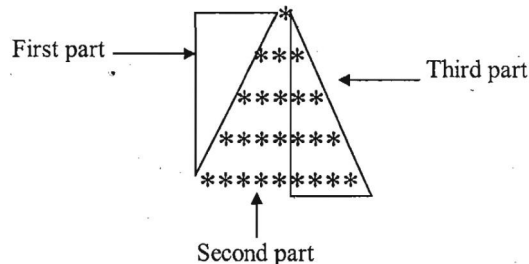
The diamond(v) can be obtained by joining pyramids(p) and (u) . So the code is also same as that of (p) and (u). Note that if n=5 then the upper pyramid has 5 lines while inverted pyramid has only 4 lines in this diamond. So the loop for the inverted pyramid here will range fro 1 to n-1 instead of 1 to n.

Program:

```

#include<stdio.h>
void main()
{
    int i,j,n;
    printf("Enter n :");
    scanf("%d",&n);
    //pyramid
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n-i;j++) //loop for number of lines in pyramid
            printf(" "); //loop for spaces(first part)
        for(j=1;j<=i;j++) //loop for second part
            printf("*");
        for(j=1;j<i;j++) //loop for third part
            printf("*"); //for next line of pyramid
        printf("\n");
    }
    //inverted pyramid
    for(i=1;i<=n-1;i++) //loop for number of lines in pyramid
    {
        for(j=1;j<=i;j++) //loop for spaces(first part)
            printf(" ");
        for(j=1;j<=n-i;j++) //loop for second part
            printf("*");
        for(j=1;j<n-i;j++) //loop for third part
            printf("*"); //for next line of pyramid
        printf("\n");
    }
}

```



Output:

```
student@student-HP-245-G6-Notebook-PC:~$ gcc v.c
student@student-HP-245-G6-Notebook-PC:~$ ./a.out
Enter n :5
  *
 ***
*****
*****
*****
*****
*****
  *
  *
  *
```