Problem Solving Using C
KCA 102: Session 2020-21

Unit-2	Lecture-9	Topic: if,if-else and nested if-else statement

Control Statements:

This deals with the various methods that C can control the *flow* of logic in a program. Control statements can be classified as un-conditional and conditional branch statements and loop or iterative statements. The Branch type includes:

1. Un-conditional:
    - goto
    - break
    - return
    - continue
2. Conditional:
    - if
    - if – else
    - Nested if
    - switch case statement
3. Loop or iterative:
    - for loop
    - while loop
    - do-while loop

Conditional Statements:

Sometimes we want a program to select an action from two or more alternatives. This requires a deviation from the basic sequential order of statement execution. Such programs must contain two or more statements that might *be* executed, but have some way to select only one of the listed options each time the program is run. This is known as conditional execution.

if statement:
Statement or set of statements can be conditionally executed using if statement. Here, logical condition is tested which, may either true or false. If the logical test is true (non zero value) the statement that immediately follows if is executed. If the logical condition is false the control transfers to the next executable statement.

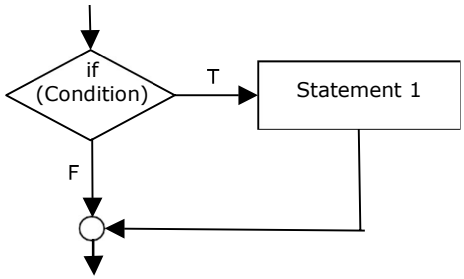
The general syntax of simple **if** statement is:

```
if (condition)
    statement_to_execute_if_condition_is_true;
```

or

```
if (condition)
{
    statement 1;
    statement 2;
     _ _ _ _;
}
```

Flowchart Segment:



if – else statement:

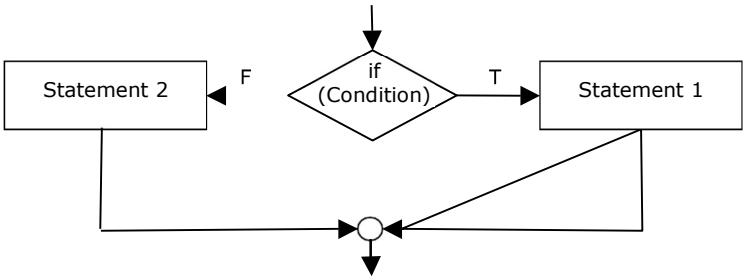
The if statement is used to execute only one action. If there are two statements to be executed alternatively, then if-else statement is used. The if-else statement is a two way branching. The general syntax of simple **if - else** statement is:

```
if (condition)
        statement_to_execute_if_condition_is_true;
else
        statement_to_execute_if_condition_is_false;
```

Where, *statement* may be a single statement, a block, or nothing, and the else statement is optional. The conditional statement produces a scalar result, i.e., an integer, character or floating point type.

It is important to remember that an if statement in C can execute only one statement on each branch (T or F). If we desire that multiple statements be executed on a branch, we must **block** them inside of a { and } pair to make them a single **compound statement**. Thus, the C code for the flowchart segment above would be:

Flowchart Segment:



Example:

```
main()
{
        int num;
        printf(" Enter a number : ");
        scanf("%d",&num);
        if (num % 2  == 0)
                printf(" Even Number ");
        else
                printf(" Odd Number ");
}
```

Nested if statement:

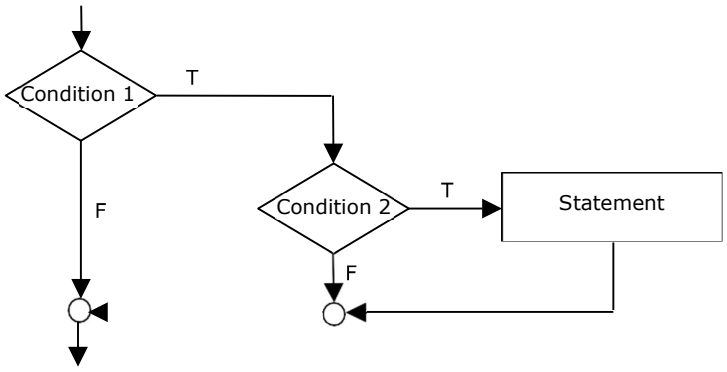
The ANSI standard specifies that 15 levels of nesting must be supported. In C, an else statement always refers to the nearest if statement in the same block and not already associated with if.

Example:

```
main()
{
        int num;
        printf(" Enter a number : ");
        scanf("%d",&num);
        if( num > 0 )
        {
                if( num % 2 == 0)
                        printf("Even Number");
                else
                        printf("Odd Number");
        }
        else
        {

                if( num < 0 )
                        printf("Negative Number");
                else
                        printf(" Number is Zero");
        }
}
```

Flowchart Segment:



if-else-if Ladder:

When faced with a situation in which a program must select from *many* processing alternatives based on the value of a single variable, an analyst must expand his or her use of the basic selection structure beyond the standard two processing branches offered by the if statement to allow for multiple branches. One solution to this is to use an approach called **nesting** in which one (or both) branch(es) of a selection contain another selection. This approach is applied to each branch of an algorithm until enough additional branches have been created to handle each alternative. The general syntax of a nested if statement is:

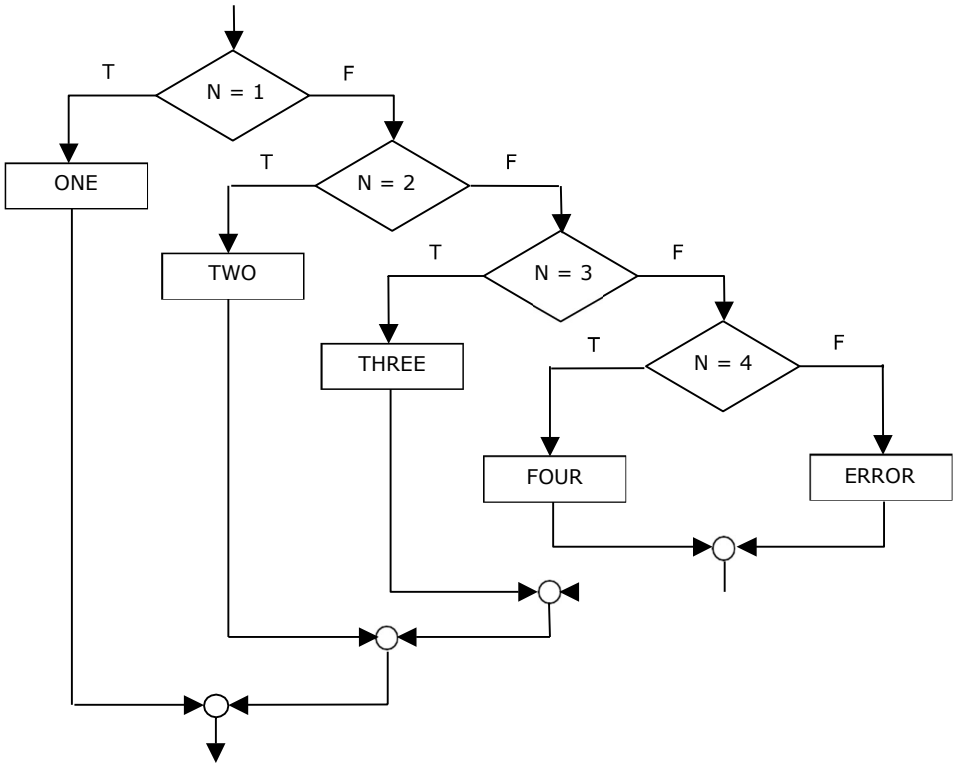
```
if (expression)
        statement1

else if (expression)
        statement2
        ..
        ..
else
        statement3
```

Example:

```
#include <stdio.h>
void main (void)
{
        int N;                          /* Menu Choice */
        printf ("MENU OF TERMS\n\n");
        printf ("1. Single\n");
        printf ("2. Double\n");
        printf ("3. Triple\n");
        printf ("4. Quadruple\n\n");
        printf ("Enter the numbe (1-4): ");
        scanf ("%d", &N);
        if (N == 1) printf ("one");
                else if (N == 2) printf ("two");
                        else if (N == 3) printf ("three");
                                else if (N == 4) printf ("four");
                                        else printf ("ERROR");

}
```



The ? : operator (ternary):

The ? (*ternary condition*) operator is a more efficient form for expressing simple if statements. It has the following form:

expression$_1$? *expression*$_2$: *expression*$_3$

It simply states as:

if *expression*$_1$ **then** *expression*$_2$ **else** *expression*$_3$

Example:

Assign the maximum of a and b to z:

```
main()
{
        int a,b,z;
        printf("\n Enter a and b ");
        scanf("%d%d",&a,&b);
        z = (a > b) ? a : b; printf("Maximum
        number: %d", z);
}
```

which is the same as:

```
if (a > b)
        z = a;
else
        z = b;
```

Problem Solving Using C
KCA 102: Session 2020-21

Unit-2	Lecture-10	Topic: switch statement, restriction of switch values and use of break statement and comparison of switch and if -else statement

The switch case statement:

The switch-case statement is used when an expression's value is to be checked against several values. If a match takes place, the appropriate action is taken. The general form of switch case statement is:

```
switch (expression)
{
        case constant1:
        statement;
        break;

        case constant2:
        statement;
        break;

        default:
        statement;
        break;
}
```

In this construct, the expression whose value is being compared may be any valid expression, including the value of a variable, an arithmetic expression, a logical comparison rarely, a bit wise expression, or the return value from a function call, but not a floating-point expression. The expression's value is checked against each of the specified cases and when a match occurs, the statements following that case are executed. When a break statement is encountered, control proceeds to the end of the switch - case statement.

The break statements inside the switch statement are optional. If the break statement is omitted, execution will continue on into the next case statements even though a match has already taken place until either a break or the end of the switch is reached.

The keyword case may only be constants, they cannot be expressions. They may be integers or characters, but not floating-point numbers or character string.

Case constants may not be repeated within a switch statement.

The last case is a special keyword default. The default statement is executed if no matches are found. The default is optional and if it is not present, no action takes place if all matches fail.

Three important things to know about switch statement:

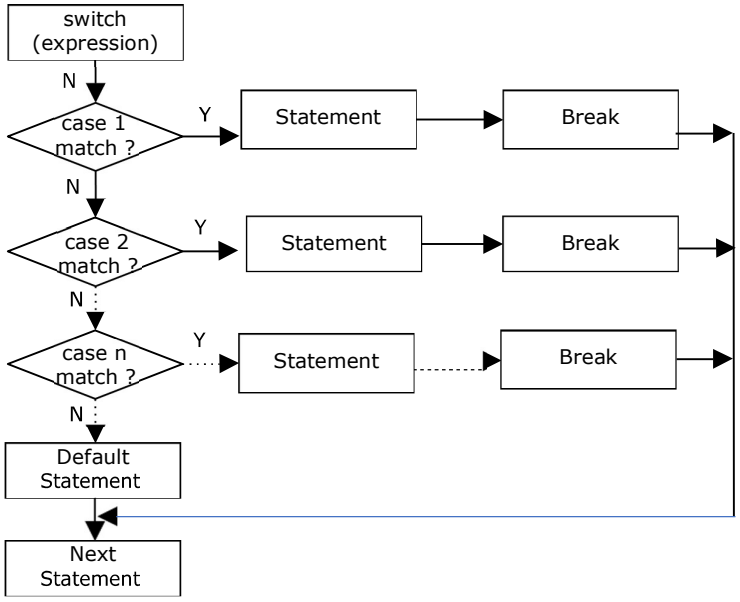
1. The switch differs from the if in that switch can only test for equality whereas if can evaluate any type of relational or logical expression.

2. No two case constants in the same switch can have identical values. But a switch statement enclosed by an outer switch may have case constants and either same.

3. If character constants are used in the switch statement, they are automatically converted to integers.

Flowchart Segment - Case Selection:

In the example below, five possible paths might be followed depending on the value stored in the character storage location X. Each path is selected based on the individual value(s) that might be stored in X.



Example 1:

```
main()
{
        char gender;
        printf ("Enter Gender code:(M/F)");
        scanf ("%c", &gender);
        switch (gender)
        {
                case 'M' : printf (" Male");
                           break;
                 case 'F' : prrintf ("Female");
                           break;
                 default : printf ("Wrong code");
        }
}
```

We can also have null statements by just including a "**;**" or let the switch statement *fall through* by omitting any statements (see *example* below).

Example 2:

```
        switch (letter)
        {
                case   `A':
                case   `E':
                case   `I' :
                case   `O':
                case  `U':
                          numberofvowels++;
                          break;
                case ' ':
                          numberofspaces++;
                          break;
                 default:
                          numberofconstants++;
                          break;
        }
```

In the above example if the value of letter is `A', `E', `I', `O' or `U' then numberofvowels is incremented. If the value of letter is ` ' then numberofspaces is incremented. If none of these is true then the default condition is executed, that is numberofconstants is incremented.

Note: In a switch maximum 255 cases can be used

Unit-2	Lecture-11	Topic: looping-for loop, while loop and do-while loop

Looping and Iteration:

Looping is a powerful programming technique through which a group of statements is executed repeatedly, until certain specified condition is satisfied. Looping is also called a repetition or iterative control mechanism.

C provides three types of loop control structures. They are:

- for statement
- while statement
- do-while statement

The for statement:

The for loop statement is useful to repeat a statement/s a known number of times. The general syntax is as follows:

```
for (initialization; condition; operation)
        statement;
```

The **initialization** is generally an assignment statement that is used to set the loop control variable.

The **condition** is an expression(relational/logical/arithmetic/bitwise ….) that determines when the loop exists.

The **Operation** defines how the loop control variable changes each time the loop is repeated.
We must separate these three major sections by semicolon.

The for loop continues to execute as long as the condition is true. Once the condition becomes false, program execution resumes on the statement following the for. The control flow of the for statement is as follows:

Example 1:

```
// printing all odd and even numbers between 1 to 5 int
        x;
        main ()
        {
                for (x=1; x <=5 ; x++)
                {
                        if( x % 2 == 0 )
                                printf( " %d is EVEN \n",x);
                        else
                                printf(" %d is ODD \n",x);
                }
        }
```

Output to the screen:
1 is ODD
2 is EVEN
3 is ODD
4 is EVEN
5 is EVEN

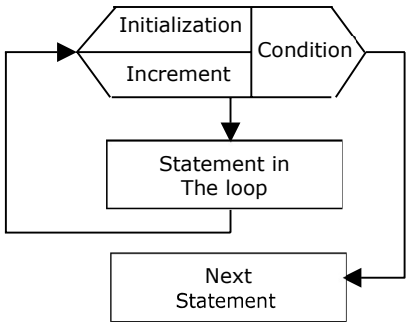
Example 2:

```
// sum the squares of all the numbers between 1 to 5 main()
{
        int x, sum = 0;
        for (x = 1; x <= 5; x ++)
        {
                sum = sum + x * x;
        }
        printf ("\n Sum of squares of all the numbers between 1 to 5 = %d ", sum);
}
```

Output to the screen:

Sum of squares of all the numbers between 1 to 5 = 55

Flowchart Segment - for Statement:



The **comma (,) operator** is used to extend the flexibility of the for loop. It allows the general form to be modified as follows:

```
for (initialization_1, initialization_2; condition; operation_1, operation_2)
    statement;
```

All the following are legal for statements in C. The practical application of such statements is not important here, we are just trying to illustrate peculiar features that may be useful:

1. for (x=0; ((x>3) && (x<9)); x++)
2. for (x=0,y=4; ((x>3) && (y<9)); x++, y+=2)
3. for (x=0, y=4, z=4000; z; z/=10)

The second example shows that multiple expressions can be separated by a , (comma).

Example:

```
main()
{
        int j ;
        double degC, degF;
        clrscr ();
        printf ("\n Table of Celsius and Fahrenheit degrees \n\n");
        printf ("Celsius Degree \t Fahrenheit Degree \n")
        degC = -20.0;
        for (j = 1; j <= 6; j++)
        {
                degC = degC + 20.0;
                degF = (degC * 9.0/5.0) + 32.0;
                printf ("\n %7.2lf\t\ %7.2lf ", degC, degF);
        }
}
```

Output:

```
Table of Celsius and Fahrenheit degrees
Celsius Degree      Fahrenheit Degree
0.00                32.00
20.00               68.00
40.00               104.00
60.00               140.00
80.00               176.00
100.00              212.00
```

Nested for loop:

Nested loops consist of one loop placed inside another loop. An example of a nested for loop is:

```
for (initialization; condition; operation)
{
        for (initialization; condition; operation)
        {
                statement;
        }
        statement;
}
```

In this example, the inner loop runs through its full range of iterations for each single iteration of the outer loop.

Example:

Program to show table of first four powers of numbers 1 to 9.

```
#include <stdio.h >

void main()
{
        int i, j, k, temp;
        printf("I\tI^2\tI^3\tI^4 \n");
        printf("-----------------------------------------\n");
        for ( i = 1; i < 10; i ++)                    /* Outer loop */
        {
                for (j = 1; j < 5; j ++)                        /* 1st level of nesting */
                {
                        temp = 1;
                        for(k = 0; k < j; k ++)
                                temp = temp * I;
                        printf ("%d\t", temp);
                }
                printf ("\n");
        }
}
```

Output to the screen:

I	I ^ 2	I ^ 3	I ^ 4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561

Infinite for loop

We can make an endless loop by leaving the conditional expression empty as given below:

```
for( ; ; )
        printf("This loop will run for ever");
```

To terminate the infinite loop the break statement can be used anywhere inside the body of the loop. A sample example is given below:

```
for(; ;)
{
        ch = getchar ();
        if(ch == 'A')
}               break;
```

```
printf("You typed an A");
```

This loop will run until the user types an A at the keyboard.

for with no bodies:
A C-statement may be empty. This means that the body of the for loop may also be empty. There need not be an expression present for any of the sections. The expressions are optional.

Example 1:

```
/*   The loop will run until the user enters 123 */

     for( x = 0; x != 123; )
          scanf ("%d", &x);
```

This means that each time the loop repeats, 'x' is tested to see if it equals 123, but no further action takes place. If you type 123, at the keyboard, however the loop condition becomes false and the loop terminates.

The initialization sometimes happens when the initial condition of the loop control variable must be computed by some complex means.

Example 2:

```
/* Program to print the name in reverse order. */

#include<conio.h>
#include<string.h>
#include<stdio.h>

void main()
{
     char s[20];
     int x; clrscr
     ();
     printf ("\nEnter your name: ");
     gets (s);
     x = strlen (s);
     for ( ; x > 0 ; )
     {
          --x;
          printf ("%c\t", s[x]);
     }
}
```

Output to the screen:

Enter your name: KIRAN

N A R I K

The while statement:

The second loop available in 'C' is while loop.

The general format of while loop is:

```
while (expression)
        statement
```

A while statement is useful to repeat a statement execution as long as a condition remains true or an error is detected. The while statement tests the condition before executing the statement.

The condition, can be any valid C languages expression including the value of a variable, a unary or binary expression, an arithmetic expression, or the return value from a function call.

The statement can be a simple or compound statement. A compound statement in a while statement appears as:

```
while (condition)
{
        statement1;
        statement2;
}
```

With the if statement, it is important that no semicolon follow the closing parenthesis, otherwise the compiler will assume the loop body consists of a single null statement. This usually results in an infinite loop because the value of the condition will not change with in the body of the loop.

Example:

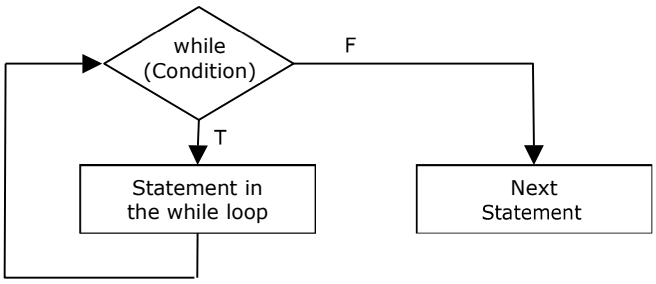
```
main()
{
        int j = 1;
        double degC, degF;
        clrscr ();
        printf ("\n Table of Celsius and Fahrenheit degrees \n\n");
        printf ("Celsius Degree \t Fahrenheit Degree \n")
        degC = -20.0;
        while (j <= 6)
        {
                degC = degC + 20.0;
                degF = (degC * 9.0/5.0) + 32.0;
                printf ("\n %7.2lf\t\ %7.2lf ", degC, degF);
                j++;
        }
}
```

Output:

Table of Celsius and Fahrenheit degrees

Celsius Degree	Fahrenheit Degree
0.00	32.00
20.00	68.00
40.00	104.00
60.00	140.00
80.00	176.00
100.00	212.00

Flowchart Segment - while Statement:



Because the while loop can accept expressions, not just conditions, the following are all legal:

```
while(x--);
while(x = x+1);
while(x += 5);
```

Using this type of expression, only when the result of x--, x=x+1, or x+=5, evaluates to 0 will the while condition fail and the loop be exited.

We can go further still and perform complete operations within the while *expression*:

```
while(i++ < 10);
```

The counts i up to 10.

```
while((ch = getchar()) !=`q')
        putchar(ch);
```

This uses C standard library functions: getchar () to reads a character from the keyboard and putchar () to writes a given char to screen. The while loop will proceed to read from the keyboard and echo characters to the screen until a 'q' character is read.

Nested while:

Example:

Program to show table of first four powers of numbers 1 to 9.

```
#include <stdio.h >

void main()
{
        int i, j, k, temp;
        printf("I\tI^2\tI^3\tI^4 \n");
        printf("-----------------------------------------\n");
        i = 1;
        while (i < 10)                                  /* Outer loop */
        {
                j = 1;
                while (j < 5)                   /* 1st level of nesting */
                {
                        temp = 1;
                        k = 1;
                        while (k < j)
                        {
                                temp = temp * i;
                                k++;
                        }
                        printf ("%d\t", temp);
                        j++;
                }
                printf ("\n");
                i++;
        }
}
```

Output to the screen:

I	I ^ 2	I ^ 3	I ^ 4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561

Unit-2	Lecture-12	Topic: nested loop with break and continue statement

The do-while statement:

The third loop available in C is do – while loop.

The general format of do-while is:

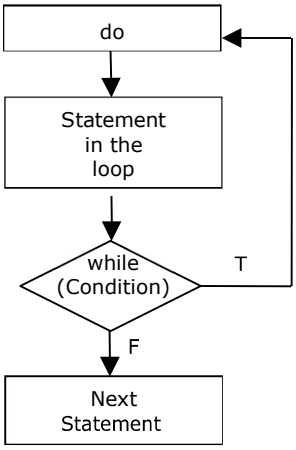
```
do
        statement;
while (expression);
```

Unlike for and while loops, which tests the condition at the top of the loop. The do – while loop checks its condition at the bottom of the loop. This means that the do – while loop always executes first and then the condition is tested. Unlike the while construction, the do – while requires a semicolon to follow the statement's conditional part.

If more than one statement is to be executed in the body of the loop, then these statements may be formed into a compound statement as follows:

```
do
{
statement1; statement2;
} while (condition);
```

Flowchart Segment of do-while Statement:



Example 1:

```
# include <stdio.h>
main()
{
        do
        {
```

```
            printf("x = %d\n", x--);
        } while(x > 0);
}
```

Output to the screen:

X = 3
X = 2
X = 1

Example 2:

```
#include <stdio.h>
void main()
{
        char ch;
        printf("T: Train\n");
        printf("C: Car\n");
        printf("S: Ship\n");
        do
        {
                printf("\nEnter your choice: ");
                fflush(stdin);
                ch = getchar();
                switch(ch)
                {
                        case 'T' :
                                        printf("\nTrain");
                                        break;
                        case 'C' :
                                        printf("\nCar");
                                        break;
                        case 'S':
                                        printf("\nShip");
                        default:        break;

                }                       printf("\n Invalid Choice");

         } while(ch == 'T' || ch == 'C' || ch == 'S');
}
```

Output to the screen:

T: Train
C: Car
S: Ship

Enter your choice: T
Train

Distinguishing between while and do-while loops:

While loop	Do-while loop
The while loop tests the condition before each iteration.	The do-while loop tests the condition after the first iteration.
If the condition fails initially the loop is skipped entirely even in the first iteration.	Even if the condition fails initially the loop is executed once.

Un-conditional (Jump) statements:

C has four jump statements to perform an unconditional branch:

- return
- goto
- break and
- continue

break statement:

We can use it to terminate a case in a switch statement and to terminate a loop.

Consider the following example where we read an integer values and process them according to the following conditions. If the value we have read is negative, we wish to print an error message and abandon the loop. If the value read is greater than 100, we wish to ignore it and continue to the next value in the data. If the value is zero, we wish to terminate the loop.

Example:

```
void main()
{
        int value;

        while (scanf("%d", &value ) == 1 && value != 0)
        {
                if(value < 0)
                {
                        printf ("Illegal value\n");
                        break;                          /* Terminate the loop */
                }
                if(value > 100)
                {
                        printf("Invalid value\n");
                        continue;                       /* Skip to start loop again */
                }
        }                                               /* end while value != 0 */
}
```

Continue statement:

The continue statement forces the next iteration of the loop to take place, skipping any code in between. But the break statement forces for termination.

Example 1:

```
/* Program to print the even numbers below 100 */

#include<stdio.h>

void main()
{
        int x;
        for(x = 1; x < 10; x++)
        {
                if (x % 2)
                        continue;
                printf ("%d\t", x)
        }
}
```

An odd number causes continue to execute and the next iteration to occur, by passing the printf () statement. A continue statement is used within a loop (i.e for, while, do – while) to end an iteration in while and do-while loops, a continue statement will cause control to go directly to the conditional test and then continue the looping process. In the case of for, first the increment part of the loop is performed, next the conditional test is executed and finally the loop continues.

Example 2:

```
main()
{
        char ch;
        while (1)
        {
                ch = getchar();
                if (ch == EOF)
                        break;
                if (iscntrl (ch))
                        continue;
                 else
                        printf ("\n not a control character");
}
```

Distinguishing between break and continue statement:

Break	Continue
Used to terminate the loops or to exist loop from a switch.	Used to transfer the control to the start of loop.
The break statement when executed causes immediate termination of loop containing it.	The continue statement when executed cause immediate termination of the current iteration of the loop.

goto statement:

goto statement provides a method of unconditional transfer control to a labeled point in the program. The goto statement requires a destination label declared as:

label:

The label is a word (permissible length is machine dependent) followed by a colon. The goto statement is formally defined as:

```
goto label;
        ‘
        ‘
        ‘
label:
        target statement
```

Since, C has a rich set of control statements and allows additional control using break and continue, there is a little need for goto. The chief concern about the goto is its tendency to render programs unreachable. Rather, it a convenience, it used wisely, can be a benefit in a narrow set of programming situation. So the usage of goto is highly discouraged.

Example:

```
Void main()
{
        int x = 6, y = 12;

        if( x == y)

                x++;
        else
                goto error;
        error:
                printf (“Fatal error; Exiting”);
}
```

The compiler doesn't require any formal declaration of the label identifiers.

Unit-2	Lecture-13	Topic: Function-Introductions, type and declaration

return statement:

A return statement is used to return from a function. A function can use this statement as a mechanism to return a value to its calling function. If now value is specified, assume that a garbage value is returned (some compilers will return 0).

The general form of return statement is:

return expression;

Where expression is any valid rvalue expression.

Example:

return x; or return(x);
return x + y or return(x + y); return
rand(x); or return(rand(x));
return 10 * rand(x); or return (10 * rand(x));

We can use as many return statements as we like within a function. However, the function will stop executing as soon as it encounters the first return. The } that ends a function also causes the function to return. It is same way as return without any specified value.

A function declared as void may not contain a return statement that specifies a value.

.

Unit-2	Lecture-14	Topic: Calling and defining functions. Passing arguments and return value

Unit-2	Lecture-15	Topic: Writing multi-function program and concept of call by value

Unit-2	Lecture-16	Topic: Recursive Functions

KIET Group of Institutions, Ghaziabad

Department of Computer Applications

(An ISO – 9001: 2015 Certified & 'A' Grade accredited Institution by NAAC)

Problem Solving Using C

KCA 102: Session 2020-21

KIET Group of Institutions, Ghaziabad

Department of Computer Applications

(An ISO – 9001: 2015 Certified & 'A' Grade accredited Institution by NAAC)

Problem Solving Using C

KCA 102: Session 2020-21

KIET Group of Institutions, Ghaziabad

Department of Computer Applications

(An ISO – 9001: 2015 Certified & 'A' Grade accredited Institution by NAAC)

Problem Solving Using C

KCA 102: Session 2020-21