UNIT-2 STATEMENTS

A ***statement*** is a part of your program that can be executed. That is, a statement specifies an action. Statements generally contain expressions and end with a semicolon.
Statements that are written individually are called ***Single/Simple statements***. Statements that are written as a block are called ***Block/Compound statements***. A block begins with an open brace { and ends with a closing brace }.
C categorizes statements into these groups:

- Selection / Branch / Decision making / Conditional statements
- Loop/ Iteration/ Repetitive statements
- Jump / Control transfer statements
- Expression Statements
- Block statements/ compound statements

SELECTION STATEMENTS:

A ***selection statement*** checks the given condition and decides the execution of statements after the success or failure of the condition. C supports two selection control statements **if** and **switch**.
1. If statement: The **if** statement is a decision making statement that allows the computer to evaluate an expression (condition) first and depending on the result of the condition i.e. true or false, it transfers the control to a particular statement.
The if statement has the following forms

a) Simple if Statement
b) if...else Statement
c) Nested if Statement
d) if..else Ladder Statement

Simple if Statement
The simple if statement contains only one condition, if the condition is true, it will execute the statements that are present between opening and closing braces. Otherwise it will not execute those statements.

Syntax:	Example:

if(condition**)**
  Single-statement
OR
if(condition**)**
{
Statement-block
}
	/* A program to check whether a number is Less than 100 or not */
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	int a;
	clrscr();
	printf("enter an integer ");
	scanf("%d",&a);
	if(a<=10)
	printf("%d is less than 100",a); getch();
	}

if...else Statement
This statement is used to define two blocks of statements in order to execute only one block. If the *condition* is true, the block of **if** is executed; otherwise, the block of **else** is executed.

Syntax:	Example:

if (condition)
{
  True Block Statements;
  ---------- 	
}
else
{
  False Block Statements;
	Write a program to check whether the given number is a positive number or a negative number
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	 int n;
	 clrscr();

---------- 	
 }
	printf("enter any number ");
	 scanf("%d",&n);
	 if(n>=0)
	 {
	 printf("\n %d is a positive number",n);
	 }
	 else
	 {
	 printf("\n %d is a negative number",n);
	 }
	 getch();
	 }

Nested if –else Statement

When an if statement is placed in another if statement or in else statement, then it is called nested if statement.

The *nested **if-else*** is used when a series of decisions are involved. In a *nested **if-else,*** an else statement always refers to the nearest **if** statement which is within the same block as the **else** and that is not already associated with an **else**.

Syntax:	Example:

if (condition)
 {
 if(condition)
 {
 True Block Statements;
 ---------- 	
 }
 }
 else
 {
 False Block Statements;
 ---------- 	
 }
	Write a program to finding greatest among three numbers
	 #include<stdio.h>
	 #include<conio.h>
	 void main()
	 {
	 int a,b;
	 clrscr();
	 printf("enter any two number ");
	 scanf("%d%d",&a,&b);
	 if(a>b)
	 {
	 if(a>c)
	 printf(" %d is greatest",a);
	 else
	 printf(" %d is greatest",c);
	 }
	
	 else
	 {
	 if(b>c)
	 printf(" %d is greatest",b);
	 else
	 printf(" %d is greatest",c);
	 }
	 getch();
	 }

if..else Ladder Statement

The ***if-else*** *ladder* is used when multipath (multiple) decisions are involved.

A multipath decision is a chain of **if-else**s in which the statement associated with each **else** is an ***if****-statement*.

Syntax:

```
if(condtion1)
{
        statements1;
}
else if(condition2)
{
        statements2;
}
else if(condition3)
{
        statements3;
}
        |
        |
        |
else if(conditionN)
{
        statementsN;
}
else
{
        statements;
}
```

Example:

Write a program to test whether the given number is a single digit or a double digit or a trible digit or more than three digits.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int n;
 clrscr();
 printf("enter any number");
 scanf("%d",&n);
 if(n>=0 && n<=9)
        printf("\n %d is a single digit number",n);
 else if(n>=10 && n<=99)
        printf("\n %d is a double digit number",n);
 else if(n>=100 && n<=999)
        printf("\n %d is a trible digit number",n);
 else
        printf("\n %d is more than three digits",n);
 getch();
 }
```

2. Switch Statement

C has a built-in multiple-branch selection statement, called **switch**, which successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

Syntax:

```
switch(expression)
        {
         case value1:
                statements1;
                break;
         case value2:
                statements2;
                break;
           |
           |
           |
         case valueN:

        statementsN;
                break;
         default:
                statements;
         }
```

Example:

Write a program to check whether the given character is a vowel or not using switch case

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char ch;
        clrscr();
        printf("enter any single character:");
        scanf("%c",&ch);
        switch(ch)
        {
         case 'a':
                printf("%c is a vowel",ch);
                break;
         case 'e':
                printf("%c is a vowel",ch);
                break;
         case 'i':
                printf("%c is a vowel",ch);
                break;
        case 'o':
```

```
                printf("%c is a vowel ",ch);
                break;
        case 'u':
                printf("%c is a vowel",ch);
                break;
        default:
                printf("\n %c is not a vowel",ch);
        }
        getch();
}
```

The ***expression*** must evaluate to an integer type. Thus, you can use character or integer values, but floating-point expressions, for example, are not allowed. The value of ***expression*** is tested against the values, one after another, of the constants specified in the **case** statements. When a match is found, the statement sequence associated with that **case** is executed until the break statement or the end of the **switch** statement is reached. 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. Technically, the **break** statements inside the switch statement are optional. They terminate the statement sequence associated with each constant. If the break statement is omitted, execution will continue on into the next case's statements until either a break or the end of the switch is reached.

Difference between if & Switch

S.No.	***switch***	***if***
1	It can test only for equality. That is, it can evaluate only integral (integer) expressions, which yield integer constants.	It can evaluate even relational or logical expressions.
2	No two ***case*** statements have identical constants in the same ***switch***.	Same conditions may be repeated for a number of times
3	Character constants are automatically converted to integers	Character constants are automatically converted to integers
4	In ***switch*** statement, nested ***if***s can be used	In nested ***if*** statement, ***switch*** can be used.

LOOP/ ITERATION STATEMENTS

In C, and all other modern programming languages, iteration statements (also called loops) allow a set of instructions to be repeatedly executed until a certain condition is reached.

Based on position of loop, loop statements are classified into two types:

1. ***entry-controlled loop (pre-test loop)- while, for***
2. ***exit-controlled loop (post-test loop)- do while***

1. While loop

While is pre tested/ entry controlled loop statement i.e first condition is checked & body of loop is executed.

Syntax:

```
initialization;
while(test condition)
{
  Statements;
  Increment / Decrement operation;
}
```

Example:

Write a program to print first 25 natural numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr();
        n=1;
        while(n<=25)
        {
```

```
                printf("%3d",n);
                n=n+1;
        }
        getch();
}
```

The ***initialization*** is an assignment statement that is used to set the loop control variable.
The ***condition*** is a relational expression that determines when the loop exits.
The ***increment/decrement*** defines how the loop control variable changes each time the loop is repeated.

2. do while loop

Do While is post tested/ exit controlled loop statement i.e first body of loop is executed & finally condition is checked. Even though condition is false, it will execute the body of loop statements at least once.

Syntax:

```
Initialization;
do
{
Statements;
 Increment/Decrement operation;
} while(test condition) ;
```

Example:

Write a program to print first 25 natural numbers

```
##include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr();
        n=1;
        do
        {
                printf("%3d",n);
                n=n+1;
        } while(n<=25);
        getch();
}
```

Difference between while/for & do while:

In while, if the condition is false, it will never execute the loop statements.
But in do-while, even though condition is false, it will execute the loop statements at least once.

While Example:

Write a program to illustrate while

```
##include<stdio.h>
#include<conio.h>
void main()
{
         int n;
         clrscr( );
         n=1;
         while(n>10)
        {
             printf("%d\n",n);
             n=n+1;
         }
         getch( );
}
```

output: No output because condition is false

Example:

Write a program to illustrate do while

```
##include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr( );
        n=1;
        do
        {
                printf("%d",n);
                n=n+1;
        } while(n>10);
        getch();
}
```

output: it prints 1 even though condition is false

3. For loop

for is pre tested/ entry controlled loop statement i.e first condition is checked & body of loop is executed.

Syntax:	Example:

for(initialization; condition; increment / decrement operation;)
{
list of statements
}
	Write a program to print first 10 numbers
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	int i;
	clrscr();
	for(i=1; i<=10; i++)
	{
	printf("%d\t",i);
	}
	getch();
	}

However, if *condition* section is omitted, the for loop becomes an endless loop, which is called an *infinite loop*. When the *condition* is absent, it is assumed to be true. The for statement may have an *initialization* and *increment/decrement* sections. But C programmers more commonly use

for (; ;)

JUMP STATEMENTS/ CONTROL TRANSFER STATEMENTS

C has four statements that perform either a conditional or unconditional branch: ***return, goto, break, and continue.*** Of these, we can use return and goto anywhere inside a function and the break and continue statements in conjunction with any of the loop statements. We can also use break with switch.

1. return Statement

A return may or may not have a value associated with it. A **return *with a value*** can be used only in a function with a non-void return type. In this case, the value associated with return becomes the return value of the function. A return ***without a value*** is used to return (exit) from a void function.

The general form of the **return** statement is

return; (OR) return *expression;*

The *expression* may be a constant, variable, or an expression. The *expression* is present only in non-**void** function. In this case, the value of *expression* will become the return value of the function. The *expression* is not present in **void** function.

2. break Statement

The break statement has two uses

a) to terminate a case in the switch statement.

b) to force immediate termination of a loop, bypassing the normal loop conditional test.

The general form of the **break** statement is break keyword followed by semicolon

break;

3. continue Statement

During the loop operations, it may be necessary to skip a part of the body of the loop under certain conditions. Like the break statement, C supports another similar statement called the ***continue*** statement. However, unlike the break which causes the loop to be terminated, the continue causes the loop to be continued with the next iteration after skipping any statements in between.

In **while** and **do-while** loops, **continue** causes the control to go to directly to the test condition and then to continue the iteration process. In the case of **for** loop, **continue** causes the control to go to the increment/decrement section of the loop and then to test condition.

The general form of the **continue** statement is continue keyword followed by semicolon

continue;

Example:
Write a program to illustrate break

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        clrscr();
        for(i=1; i<=10; i++)
        {
              if(i==6)
              break;
               printf("%d\t",i);
        }
        getch();
}
```

output: 1 2 3 4 5

Example:
Write a program to illustrate continue

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        clrscr();
        for(i=1; i<=10; i++)
        {
              if(i==6)
              continue;
              printf("%d\t",i);
        }
        getch();
}
```

output: 1 2 3 4 5 7 8 9 10

4. *goto* Statement

The goto statement is used to branch from one point (statement) to another in the program.

The **goto** statement requires a *label* in order to identify the place where the branch is to be made. A ***label*** is a valid identifier followed by a colon. The *label* is placed immediately before the statement where the control is to be transferred. Furthermore, the label must be in the same function as the **goto** that uses it – we can't jump between functions.

The general form of goto is

label:		goto label;
--		----
--	(OR)	----
goto label;		label:

The **goto** breaks the normal sequential execution of the program.

If the *label* is before the **goto** statement, a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a ***backward jump***. If the *label* is placed after the **goto** statement, some statements will be skipped and the jump is known as a ***forward jump***.

Example:
Write a program to illustrate goto statement

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        clrscr();
        for(i=1; i<=10; i++)
        {
              if(i==6)
              goto xyz;
               printf("%d\t",i);
        }
        xyz: printf("\nthankyou\n");
        getch();
}
```

output: 1 2 3 4 5
thankyou