MODULE 2

1. Define operator. List different types of operators. Explain any four operators with suitable example.

An operator is a symbol that specifies the mathematical, logical, or relational operation to be performed. The different types of operators supported by C are:

- Arithmetic operators
- Relational operators
- Equality operators
- Logical operators
- Unary operators
- Shift operators
- Conditional operators
- Bitwise operators
- Assignment operators
- Comma operator

Arithmetic operators: Arithmetic operators are used to perform various arithmetic operations on operands. Table lists various arithmetic operators, their syntax & usage:

Operation	Operator	Syntax	Comment	Result
Multiply	*	a * b	Result = a * b (a=5, b=4)	20
Divide	/	a / b	Result = a / b (a=9, b=3)	3
Addition	+	a + b	Result = a + b (a=4, b=2)	6
Subtraction	-	a - b	Result = a – b (a=8, b=1)	7
Modulus	%	a % b	Result = a % b (a=12, b=5)	2

Equality operators: C supports two kinds of equality operators to compare their operands for strict equality or inequality. They are:

- equal to (==) operator
- not equal to (!=) operator

Table below summarizes equality operators:

Operator	Meaning
==	Returns 1 if operands on both the side of operators are equal, 0 otherwise
!=	Returns 1 if both operands do not have the same value, 0 otherwise

Unary operators: Unary operators act on single operands. C supports three unary operators: Unary minus, increment and decrement operators.

- Increment operator (++) and decrement operator (- -)

The increment operator increases the value of its operand by 1 while the decrement operator decreases the value of its operand by 1.

For example, x++ is equal to x = x + 1 and --x is equal to x = x – 1.

The increment / decrement operator have two variants: **prefix** and **postfix**.

In a prefix expression (++x or --x) the operator is applied before an operand is fetched for computation and thus altered value is used for the computation of the expression.

Example: int x=10, y;

```
y = ++x;
```

is equivalent to writing

```
x = x + 1;
y = x;
```

In a postfix expression (x++ or x--) the operator is applied after an operand is fetched for computation and thus unaltered value is used for the computation of the expression.

Example: int x=10, y;

```
y = x++;
```

is equivalent to writing

```
y = x;
x = x + 1;
```

- Unary Minus (-): When an operand is preceded by a minus sign, the unary operator negates its value.

For example: int a, b=10;

```
a = - (b);
```
then, the value of a will be -10.

Shift operators: C supports two bitwise shift operators: shift-left (<<) and shift-right(>>). These operators shifts the bits either to the right or to the left. The syntax is given as:

```
operand operator num;
```

here, the bits in the operand are shifted to left or right as specified by the operator by the number of places denoted by num.

Example: if x = 0011 0101, then

x << 2 produces 1101 0100; here every bit in x is shifted to left by 2 place. So, for every shift MSB of x is lost and the LSB of x is set to 0.

Similarly, if x = 0011 0101, then

x >> 2 produces 0000 1101; here every bit in x is shifted to right by 2 place. So, for every shift LSB of x is lost and the MSB of x is set to 0.

2. Explain Logical and relation operators with appropriate examples.

C supports three logical operators: Logical AND (&&), Logical OR (||) and logical NOT (!).

Logical AND: Logical AND operator is a binary operator which simultaneously evaluates two values or relational expressions. It returns a true value if both operands are true. If both or one of the operands is false, then it returns a false value.

Example: (a < b) && (b > c); // This statement returns true value if both the expressions are true i.e. if b is greater than both a and c.

Logical AND: Logical OR operator is a binary operator which simultaneously evaluates two values or relational expressions. It returns a false value if both operands are false. If both or one of the operands is true, then it returns a true value.

Example: (a < b) && (b > c); // This statement returns true value if any one expression or both the expressions are true i.e. if either b is greater than a or b is greater than c or b is greater than both a and c.

Logical NOT: Logical NOT operator takes a single expression and negates the value of expression. It produces a 0 if the expression evaluates to non-zero value and produces a 1 if the expression produces a zero.

Example: int a =10, b; b = !a; // Here the value of b = 0, because value of a = 10 (non- zero value). !a = 0.

The truth table of Logical AND, Logical OR and Logical NOT is given in below table:

Operands		Logical AND	Logical OR	Operand	Logical NOT
A	B	A && B	A || B	A	!A
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1		
1	1	1	1		

Relational operator: The relational operator, also known as comparison operator, is an operator compares two values. Relational operator returns true or false value, depending on whether the conditional relationship between two operands holds or not.

Table below lists various relational operators.

Operator	Meaning	Example
<	Less than	3 < 5 gives 1
>	Greater than	7 > 9 gives 0
<=	Less than or equal	6 <= 5 gives 0
>=	Greater than or equal	8 >= 8 gives 1

3. Explain conditional operator. Give example. (Write a program to find the largest of three numbers using ternary operator).

The conditional or ternary operator (? :) is like an if-else statement that can be used within the expression. The syntax of conditional operator is:

exp1 ? exp2 : exp3;

here, exp1 is evaluated first. If it is true, then exp2 is evaluated and becomes the result of the expression, otherwise exp3 is evaluated and becomes the result of the expression.

Example: large = (a > b) ? a : b;

Here, a>b is evaluated first. If a is greater than b, then large = a, else large = b.

```
#include<stdio.h>
int main( )
{
int n1, n2, n3, largest;
printf ( “Enter first number” );
scanf ( “%d”, &n1 );
printf ( “Enter second number” );
scanf ( “%d”, &n2 );
printf ( “Enter third number” );
scanf ( “%d”, &n3 );
largest = n1 > n2 ? ( n1 > n3 ? n1 : n3) : ( n2 > n3 ? n2 : n3)
printf ( “The largest number is %d”, largest );
return 0;              }
```

4. What is type conversion and typecasting? Explain.

When expressions involve two different data types, like multiplying a floating point number and an integer, then situations are handled either through type conversion or typecasting.

Type conversion or typecasting of variables refers to changing a variable of one data type into another. Type conversion is done implicitly, whereas typecasting has to be done explicitly by the programmer.

Type Conversion: Type conversion is done when the expression has variables of different data types. To evaluate the expression, the data type is promoted from lower to higher level where the hierarchy of data types (from higher to lower) can be given as: double, float, long, int, short, and char. Figure 9.14 shows the conversion hierarchy of data types.

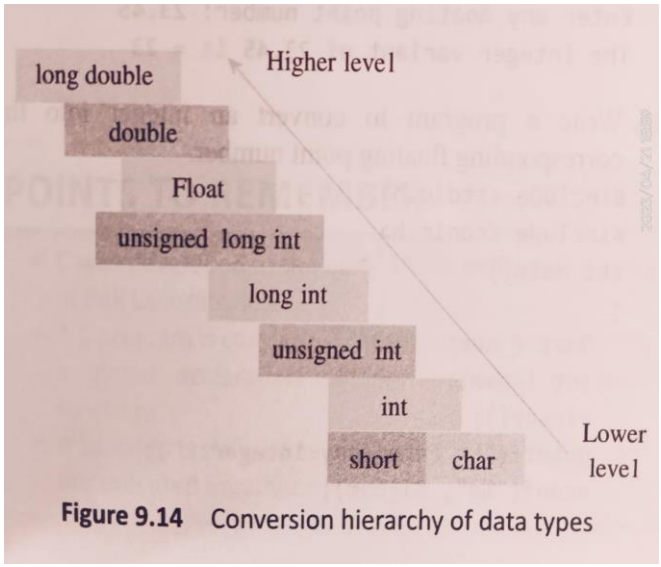


Figure 9.14 Conversion hierarchy of data types

Type conversion is automatically done when we assign an integer value to a floating point variable. Consider the code given below in which an integer data type is promoted to float. This is known as promotion (when a lower level data type is promoted to a higher type).

```
float x;
int y = 3;
x = y;
```

Now x = 3.0, as automatically integer value is converted into its equivalent floating point representation.

Typecasting: Typecasting is also known as forced conversion. It is done when the value of a higher data type has to be converted into the value of a lower data type. This casting is under the programmer's control and not under compiler's control.

For example, if we need to explicitly typecast a floating point variable into an integer variable, then the code to perform typecasting can be given as:

```
float salary 10000.00;
int sal;
sal = (int) salary;
```

When floating point numbers are converted to integers, the digits after the decimal are truncated.

5. Why conditional branching statements are required in C program? List different types and explain any two.

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:

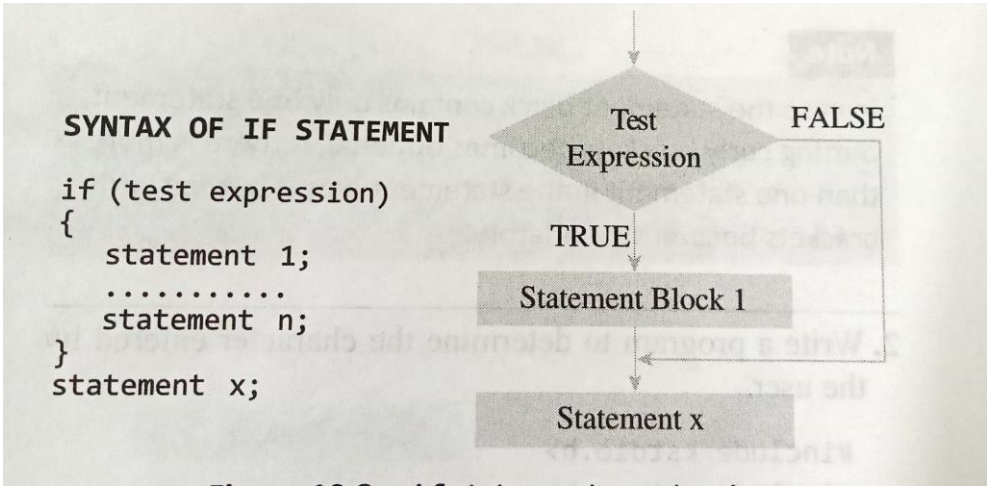
- if statement
- if-else statement
- if-else-if statement
- switch statement

if statement: The syntax and flow diagram of if statement is shown in figure.

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

Example: Program to check whether the number is even or odd.

```
#include<stdio.h>
void main( )
{
int x = 10;
if ( x > 0)
printf ( “Positive number” );
}
```



Output: Positive number

if-else statement: The syntax and flow diagram of if-else statement is shown in figure.

The statement block may include one or more statements. First, the test expression is evaluated. If the expression is true, the statement block1 is executed and statement block2 is skipped. Otherwise, if the expression is false, the statement block2 is executed and statement block1 is skipped. After executing statement block1 or 2, the control will pass to statement x.

Example: Program to check whether the number is even or odd.

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

```
int num;
printf ( “Enter any number” );
scanf (“ %d”, &num );
if ( num % 2 ==0 )
printf ( “The number is even” );
else
printf ( “The number is odd” );
}
```

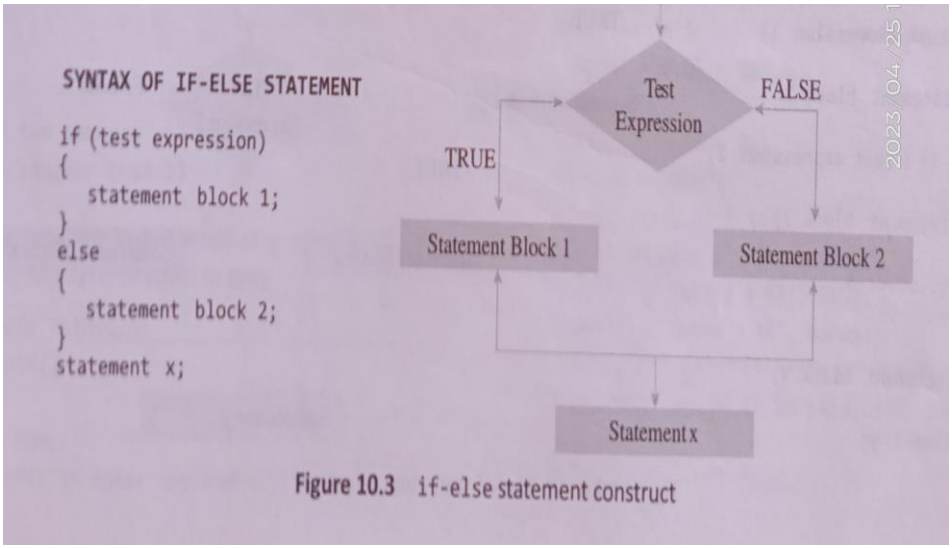


Figure 10.3 if-else statement construct

6. Explain with syntax and flowchart: if-else-if statement (nested-if).

The if-else-if construct is also known as nested if construct. Figure shows the syntax and flowchart of if-else-if statement.

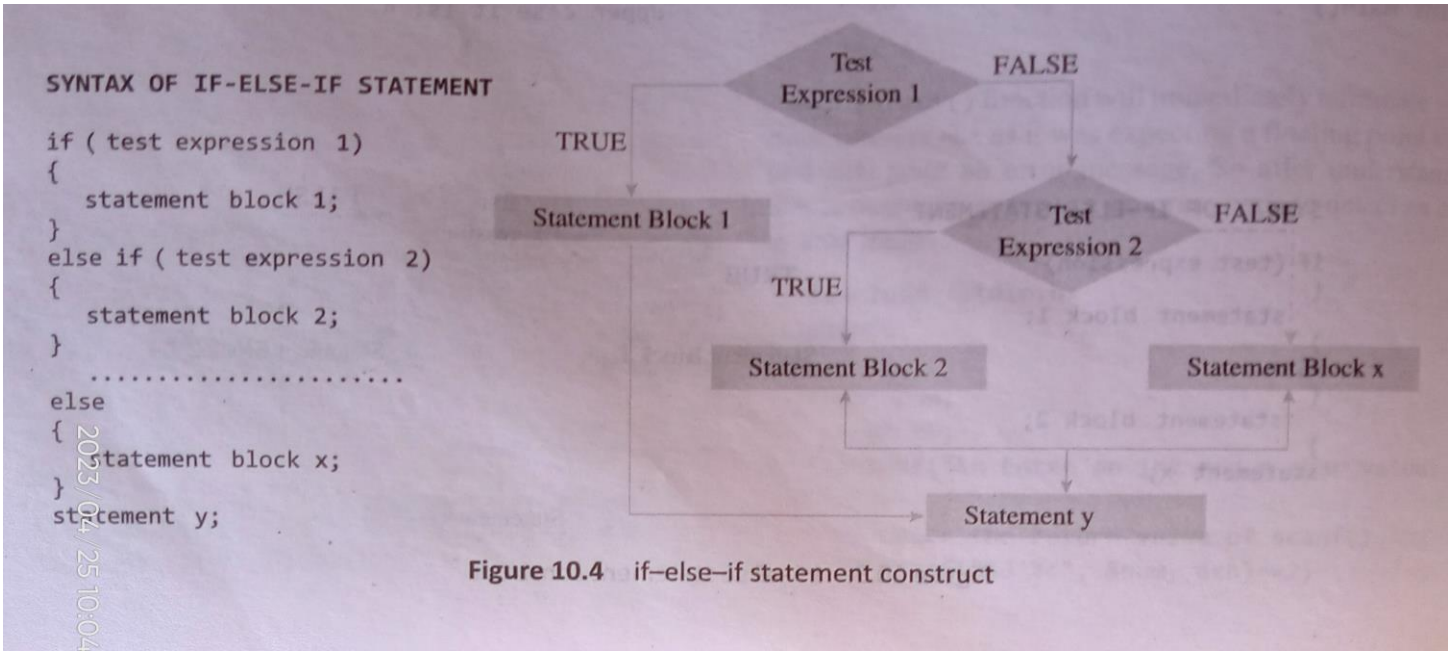


Figure 10.4 if–else–if statement construct

First, the test expression1 is evaluated. If it is true, the statement block1 is executed. If it is false, then the test expression2 will be evaluated. If it is true, the statement block2 is executed, otherwise the statement block-x is executed. After executing any of the statement blocks under the test expressions, the control will pass to statement y.

Example: Program to check compare two numbers.

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

```
int a, b;
printf ( “Enter two numbers” );
scanf (“ %d %d”, &a, &b );
if ( a == b )
printf ( “The numbers are equal” );
if ( a > b )
printf ( “ %d is greater than %d”, a, b );
else
printf ( “ %d is smaller than %d”, a, b );
}
```

7. Explain switch statement with syntax and an example.

A switch case statement is a multi-way decision statement that is simplified version of if-else-if statement that evaluates only one variable. The syntax and flow diagram of switch statement is shown in figure.

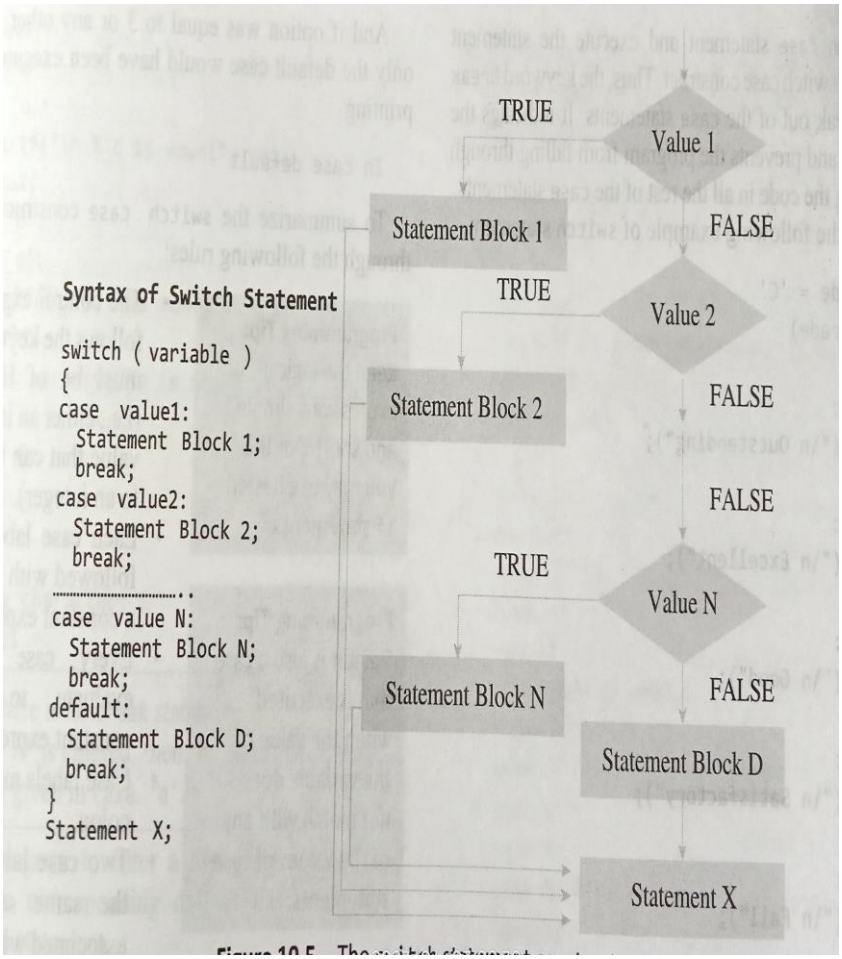
As shown in figure, the statement blocks refer to statement lists that may contain zero or more statements. The switch case statement compares the value of variable given in the switch statement with value of each case statement that follows. When the value of switch and the case statement matches, the statement block of that particular case is executed.

The default is also a case that is executed when the value of variable does not match with any of the values of the case statement. The break statement must be used at the end of each case, which tells the compiler to jump out of the switch case statement and execute the statement following the switch construct.

Example: Program to determine whether an entered character is Vowel or not.

```
#include<stdio.h>
int main( )
{
char ch;
printf(“Enter any character”);
scanf(“%c”, &ch);
switch(ch)
```

```
{
case 'A':
case 'a': printf("\n %c is a vowel", ch);
        break;
case 'E':
case 'e': printf("\n %c is a vowel", ch);
        break;
case 'I':
case 'i': printf("\n %c is a vowel", ch);
        break;
case 'O':
case 'o': printf("\n %c is a vowel", ch);
        break;
case 'U':
case 'u': printf("\n %c is a vowel", ch);
        break;
default: printf("\n %c is not a vowel", ch);
        break;
}
return 0;
}
```



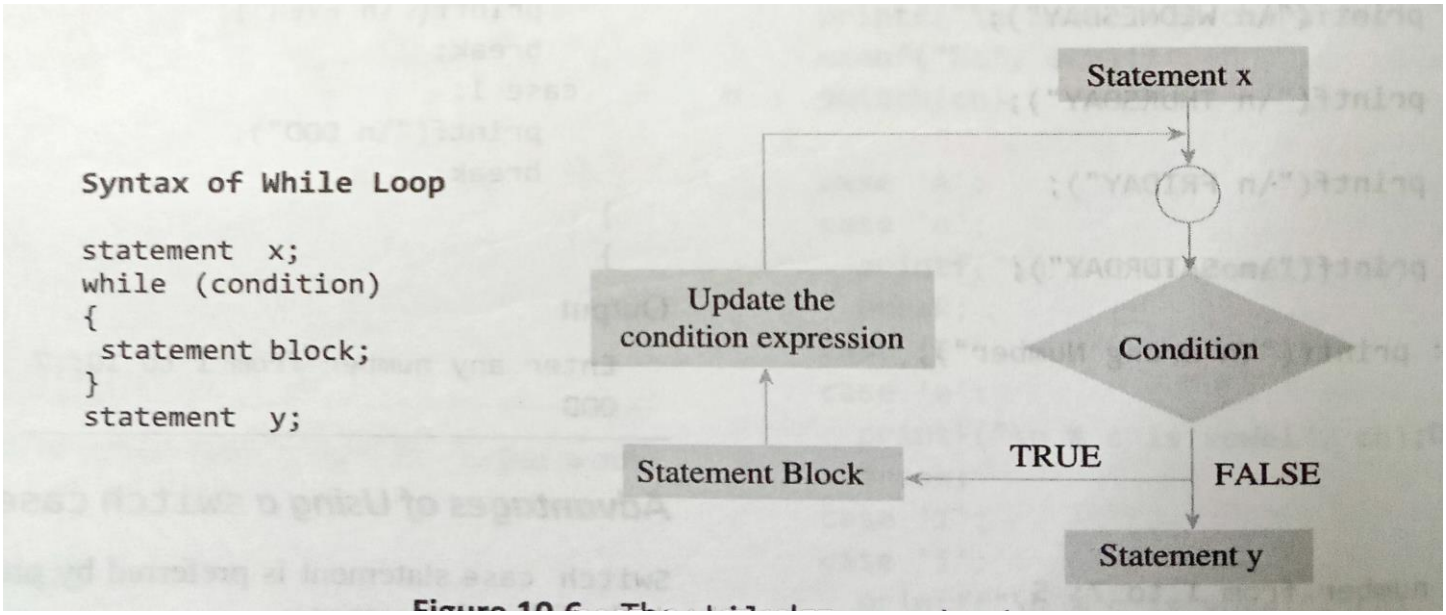
8. Why iterative (looping) statements are required in C program? List different types and explain with example.

Iterative statements are used to repeat the expression of a list of statements, depending on the value of an integer expression. C language supports three types of iterative statements also known as looping statements. They are:

- while loop
- do-while loop
- for loop.

while loop: The while loop is used to repeat one or more statements while a particular condition is true. The syntax and flowchart for while loop is shown in figure.

In while loop, the condition is checked before any of the statements in the statement block is executed. If the condition is true, then only the statements will be executed otherwise if the condition is false, the control will jump to statement y, which is the immediate statement outside the while loop block. The while loop will execute the statements inside the loop repeatedly as long as condition is true.



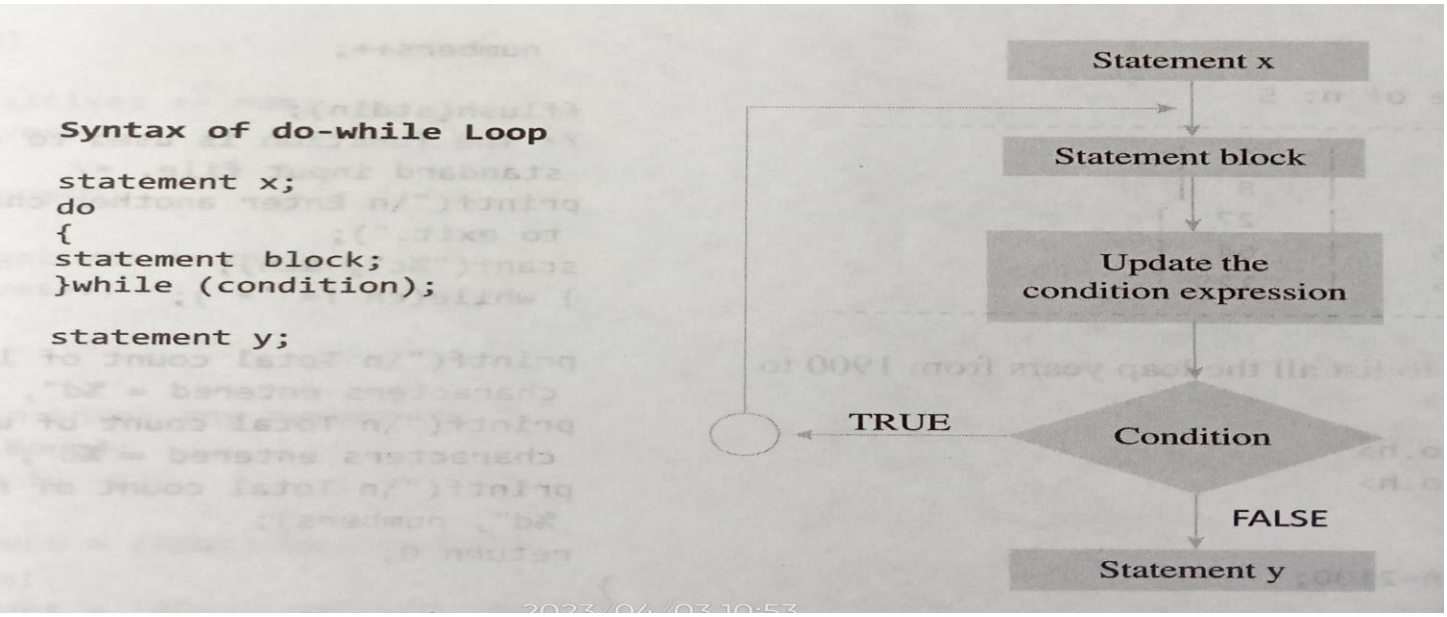
Example: Program to calculate sum of first 10 numbers.

```
#include<stdio.h>
int main( )
{
int i=1, sum=0;
while ( i <= 10 )
{
sum = sum + i;
i = i + 1;
}
printf( " The sum of first 10 numbers is %d", sum );
return 0;
}
```

do-while loop: The do-while loop is similar to while loop. The only difference is that in a do-while loop, the condition is evaluated at the end of the loop. This means that, the statements in the body of the loop will be executed at least once even if the condition is true. The syntax and flowchart for a do-while loop is shown in figure.

Like while loop, the do-while loop continues to execute the statements inside the loop repeatedly as long as condition is true.

A do-while loop is referred to as bottom-checking loop since control condition is placed on the last line of the code.

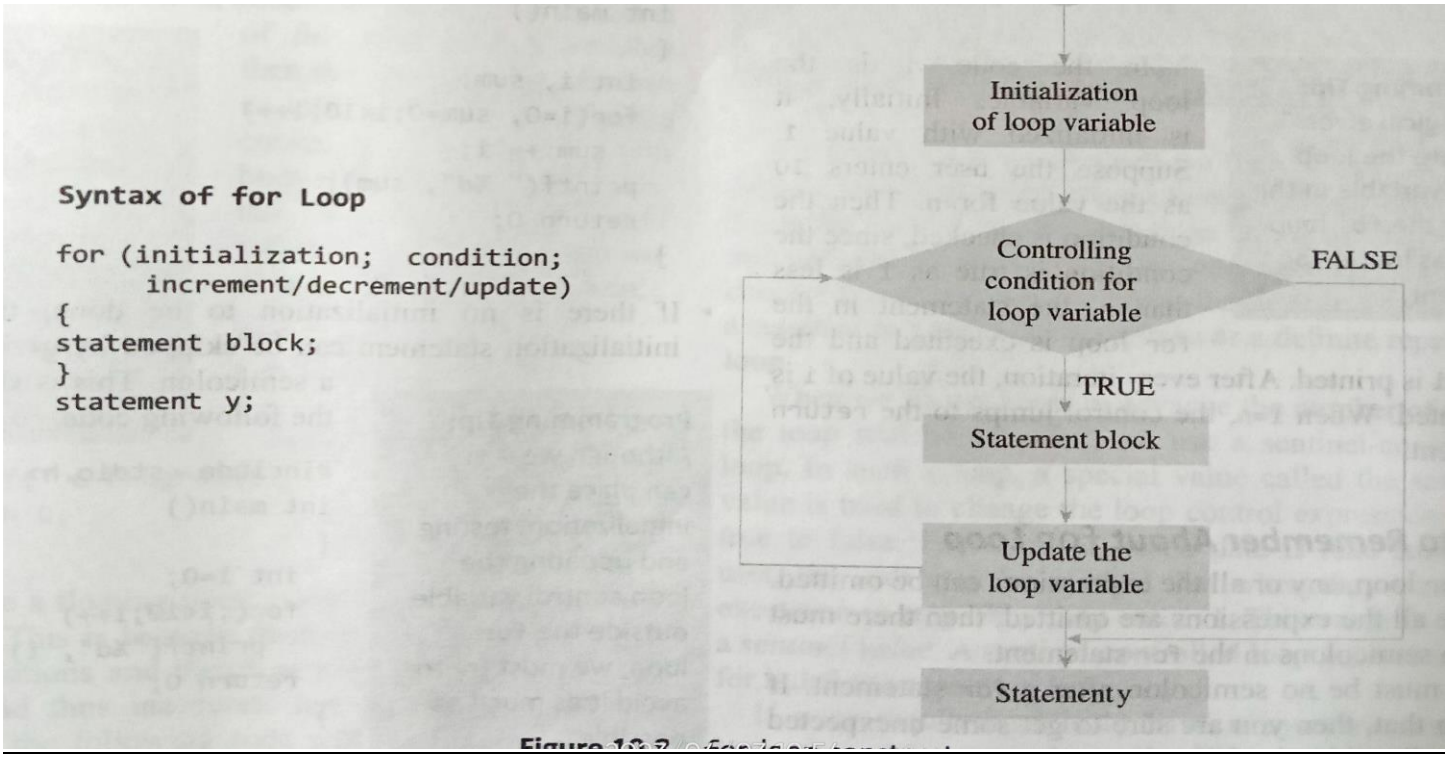


Example: Program to calculate sum of first 10 numbers.

```
#include<stdio.h>
void main( )
{
int i=1, sum=0;
do
{
sum = sum + i;
i = i + 1;
} while ( i <= 10 );
printf( " The sum of first 10 numbers is %d", sum );
}
```

for loop: Like while and do-while loop, the for loop provides a mechanism to repeat a task until a particular condition is true. The for loop is usually known as definite or determine loop because the programmer will know exactly how many times the loop will repeat. The syntax and flowchart for a do-while loop is shown in figure.

In for loop, the loop variable is initialized only once. With every iteration of the loop, the value of the loop variable is updated and the condition is checked. If the condition is true, then the statement block of the loop is executed, else the statements comprising the statement block of the loop are skipped and the control jumps to the immediate statement following the for loop body.



Example: Program to calculate sum of first 10 numbers.

```
#include<stdio.h>
void main( )
{
int i, sum=0;
for ( i=1; i<=10; i++ )
sum = sum + i;
printf( " The sum of first 10 numbers is %d", sum );
}
```

9. Explain break and continue statement with example.

break statement: The break statement is used to terminate the execution of the nearest enclosing loop in which it appears. When the compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.

Syntax: break;

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

Output: 1 2 3 4

continue statement: The continue statement is used in the body of the loop.

When the compiler encounters a continue statement then rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

Syntax: continue;

```
#include<stdio.h>
int main( )
{
int i;
for ( i =0; i <= 10; i++)
{
```

```
if ( i == 5 )
continue;
printf (“ \t %d”, i);
}
return 0;
}
```

Output: 1 2 3 4 6 7 8 9 10

10. Compare and contrast do – while and while loops with an example.

	while	**do-while**
1.	Condition is checked first then statement(s) is executed.	Statement(s) is executed at least once, then condition is checked.
2.	No semicolon at the end of while: while(condition)	Semicolon at the end of while: while(condition);
3.	If there is a single statement, brackets are not required.	Brackets are always required.
4.	Statement(s) can be executed zero times if the condition is false.	The statement(s) is executed at least once even if the condition is false.
5.	Variable in condition is initialized before the execution of loop.	Variable may be initialized before or within the loop.
6.	It is an entry controlled loop.	It is an exit controlled loop.

7.	Syntax:
	while(condition)
	{
	statement(s); //body of the loop
	}
		Syntax:
		do
		{
		statement(s); // body of the loop
		}while(condition);

8.	Example: Sum of first 10 numbers.
	int main ( )
	{
	int i, sum=0;
	while(i <=10)
	{
	sum = sum + i ;
	i = i + 1;
	}
		Example: Sum of first 10 numbers.
		int main ( )
		{
		int i, sum=0;
		do
		{
		sum = sum + i ;
		i = i + 1;
		} while(i <=10);

11. Explain goto statement.

goto statement: The goto statement is used to transfer the control to a specified label. The label must reside in the same function and can appear only before one statement in the same function.

The syntax is shown in figure.

Fig: goto statement

Here, label is an identifier that specifies the place where the branch is to be made. Label can be any valid variable name that is followed by a colon (:). The label is placed immediately before the statement where the control has to be transferred.

The label can be placed anywhere in the program either before or after the goto statement as shown. If the label is placed after the goto statement, then it is called a forward jump and in case it is located before the goto statement, it is said to be a backward jump.

The goto statement is often combined with the if statement to cause a conditional transfer of control.

IF condition **THEN goto** Label

12. Write a C Program to check the given character is Lowercase or Uppercase or number or Special Character.

```
#include<stdio.h>
int main()
{
    char c;      //for initialize of character
      printf("Enter any character : ");          //to take user input
      scanf("%c", &c);
         if(c>='A' && c<='Z')     //to find true of upper case value.
         printf("Character is an upper case");
     else if(c>='a' && c<='z')            //to check of lowercase character
        printf("Character is a lower case");
    else if(c>='0'&& c<='9')              //to check it is not a character
       printf("It is not a character");
```

```
        else                                    //all condition false, then
         printf("Character is a special character");
  return 0;
}
```

13. Write a C Program to find Mechanical Energy of a particle using $E = mgh+1/2mv^2$.

```
#include <stdio.h>
int main( )
{
float m, h, v, p, k, e; // Declaring the variables
printf( "Enter Mass of the body:\n" ); // User input for mass
scanf( "%f", &m );
printf( "Enter displacement of the body:\n" ); // User input height
scanf( "%f", &h );
printf( "Enter velocity of the body:\n" ); // User input for velocity
scanf("%f", &v );
p=m*9.8*h; //To calculate Potential energy
k=0.5*m*(v*v); //To calculate Kinetic energy
e=p+k; //To calculate Mechanical energy
printf( "Potential energy of the body = %f Joules \n", p );
printf( "Kinetic energy of the body = %f Joules \n", k );
printf( "Mechanical energy of the body = %f Joules \n", e );
return 0;
}
```

14. Develop a program to convert an integer (floating point) into the corresponding floating point number (integer) using Type casting.

Integer to floating point number	**Floating point number to integer**
#include<stdio.h> int main( )	#include<stdio.h> int main( )

```
{
int i_num;
float f_num;
printf ( “Enter any integer number” );
scanf ( “%d”, &i_num );
f_num = ( float )i_num;
printf ( “ The floating point variant of %d is = %f”, i_num, f_num );
return 0;
}
```

```
{
float f_num;
int i_num;
printf ( “Enter any floating point number” );
scanf ( “%f”, &f_num );
i_num = ( int )f_num;
printf ( “ The integer variant of %f is = %d”, f_num, i_num );
return 0;
}
```

15. Develop a program to print below pattern.

```
$$$$$$
$$$$$$
$$$$$$
$$$$$$
$$$$$$
```

```
$
$$
$$$
$$$$
$$$$$
```

```
#include<stdio.h>
int main( )
{
int i, j;
for ( i =1; i <= 5; i++) // 5 rows
{
printf( “\n“ );
for ( j =1; j <= 6; j++) // 6 symbols in each row
printf( “$“ );
}
return 0;
}
```

```
#include<stdio.h>
int main( )
{
int i, j;
for ( i =1; i <= 5; i++)
{
printf( “\n“ );
for ( j =1; j <= i; j++)
printf( “$“ );
}
return 0;
}
```

16. Develop a program to print following pattern.

1 12 123 1234 12345 123456	1 22 333 4444 55555 666666

#include<stdio.h>
int main( )
{
int i, j;
for (i =1; i <= 6; i++) // 6 rows
{
printf("\n");
for (j =1; j <= i; j++)
printf("%d", **j**);
}
return 0;
}
	#include<stdio.h>
	int main( )
	{
	int i, j;
	for (i =1; i <= 6; i++)
	{
	printf("\n");
	for (j =1; j <= i; j++)
	printf("%d", **i**);
	}
	return 0;
	}

17. Develop a program to sum the following series

1 + 1/2 + 1/3 + ¼ +.....+ 1/n	$1/1^2 + 1/2^2 + 1/3^2 + 1/2^2 + \ldots\ldots\ldots\ldots$

#include<stdio.h>
int main( )
{
float n, i, a, sum=0.0;
printf(" Enter the value of n");
scanf ("%f", &n);
for (i =1; i <= n; i++) //
{
a = 1/i;
sum = sum + a;
}
printf("The sum of series is %f", sum);
return 0;
}
	#include<stdio.h>
	int main( )
	{
	float n, i, a, sum=0.0;
	printf(" Enter the value of n");
	scanf ("%f", &n);
	for (i =1; i <= n; i++) //
	{
	a = 1/pow(i,2);
	sum = sum + a;
	}
	printf("The sum of series is %f", sum);
	return 0;
	}