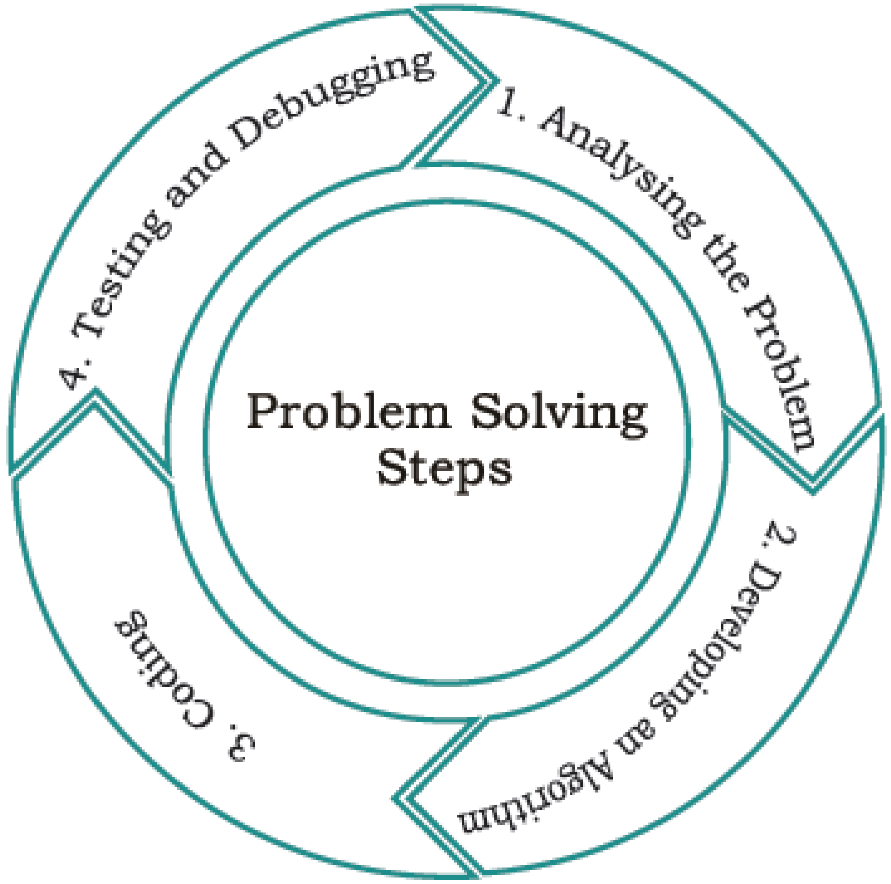
Computer Science

Chapter 2

Concepts of Algorithm and Program

I. Problem solving

It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving.



II. ALGORITHM

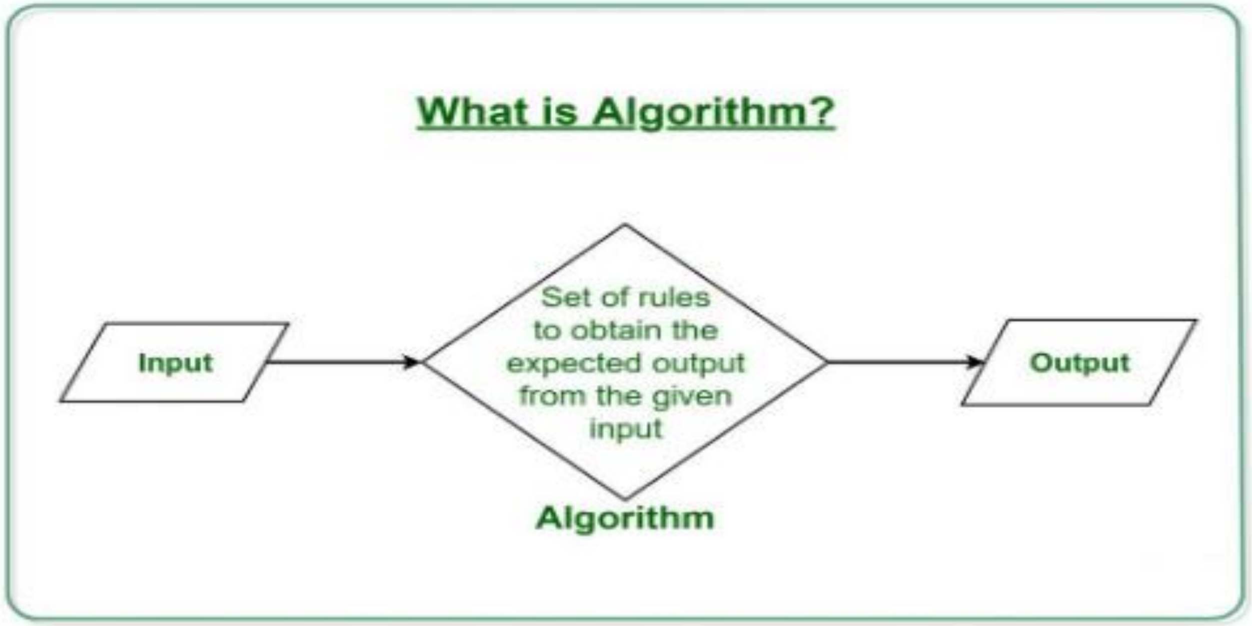
The word “algorithm” relates to the name of the mathematician Al-khowarizmi, which means a procedure or a technique. Software Engineer commonly uses an algorithm for planning and solving the problems. An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

An algorithm consists of a structured sequence of instructions

II. ALGORITHM

Algorithm has the following characteristics

- **Input**: An algorithm may or may not require input
- **Output:** Each algorithm is expected to produce at least one result
- **Definiteness**: Each instruction must be clear and unambiguous
- **Finiteness**: If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

The algorithm should be written in such a way that, it can be used in similar programming languages.

II. ALGORITHM

HOW TO WRITE ALGORITHMS?

Step 1 → **Define your algorithms input:** Many algorithms take in data to be processed, e.g. to calculate the area of rectangle input may be the rectangle height and rectangle width.

Step 2 → **Define the variables:** Algorithm's variables allow you to use it for more than one place. We can define two variables for rectangle height and rectangle width as HEIGHT and WIDTH (or H & W). We should use meaningful variable name e.g. instead of using H & W use HEIGHT and WIDTH as variable name.

Step 3 → **Outline the algorithm's operations:** Use input variable for computation purpose, e.g. to find area of rectangle multiply the HEIGHT and WIDTH variable and store the value in new variable (say) AREA. An algorithm's operations can take the form of multiple steps and even branch, depending on the value of the input variables.

Step 4 → **Output the results of your algorithm's operations:** In case of area of rectangle output will be the value stored in variable AREA. if the input variables described a rectangle with a HEIGHT of 2 and a WIDTH of 3, the algorithm would output the value of 6.

II. ALGORITHM

Examples of Algorithms in Programming

Example 1:

Write an algorithm to add two numbers entered by user

Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
Sum = num1+num2
Step 5: Display sum
Step 6: Stop

II. ALGORITHM

Examples of Algorithms in Programming

Example 2:

Write an algorithm to find the largest among three different numbers entered by user.

```
Step 1: Start
Step 2: Declare variables a, b and c.
Step 3: Read variables a, b and c.
Step 4: If a>b
If a>c
Display a is the largest number.
Else
Display c is the largest number.
Else
If b>c
Display b is the largest number.
Else
Display c is the greatest number.
Step 5: Stop
```

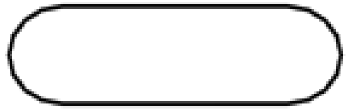
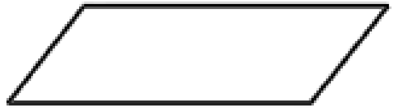
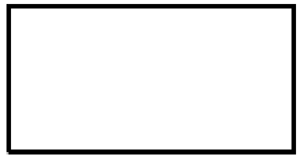
III. Representation of Algorithms: flowchart

- The flowchart is a diagram which visually presents the flow of data through processing systems. This means by seeing a flow chart one can know the operations performed and the sequence of these operations in a system. Algorithms are nothing but sequence of steps for solving problems. So a flow chart can be used for representing an algorithm. A flowchart, will describe the operations (and in what sequence) are required to solve a given problem. You can see a flow chart as a blueprint of a design you have made for solving a problem.

- A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows. Each shape represents a step of the solution process and the arrow represents the order or link among the steps.

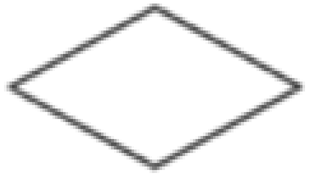
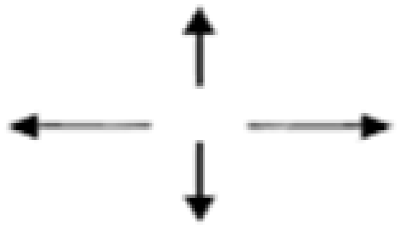
III. Representation of Algorithms: flowchart

- There are standardised symbols to draw flowcharts

Symbol Name	Symbol	function
Oval		Used to represent start and end of flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations

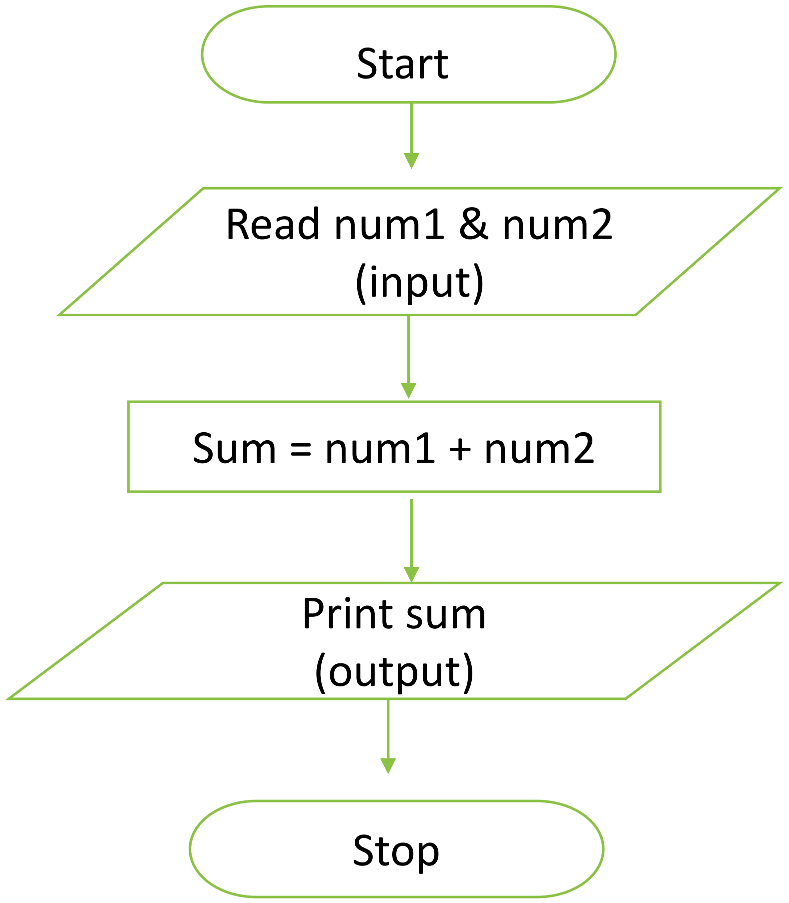
III. Representation of Algorithms: flowchart

- There are standardised symbols to draw flowcharts

Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols

III. Representation of Algorithms: flowchart

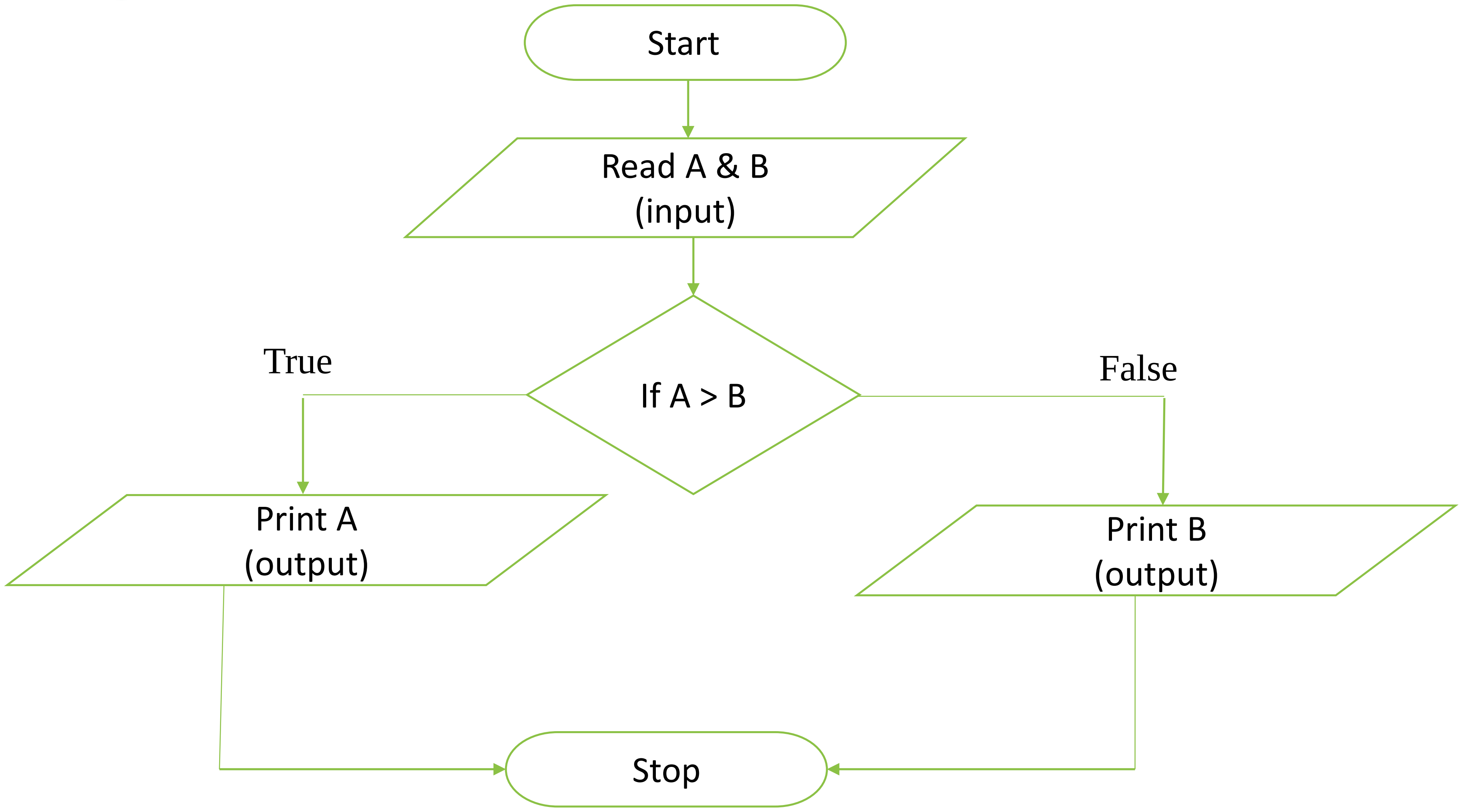
Example 1: Flowchart for an algorithm which gets two numbers and prints sum of their value



III. Representation of Algorithms: flowchart

Example 2:

Flowchart for an algorithm find the greater number between two numbers



IV. Coding

- Once an algorithm is finalised, it should be coded in a high-level programming language as selected by the programmer. The ordered set of instructions are written in that programming language by following its syntax. Syntax is the set of rules or grammar that governs the formulation of the statements in the language, such as spellings, order of words, punctuation, etc.

IV. Coding

- The machine language or low level language consisting of 0’s and 1’s only is the ideal way to write a computer program. Programs written using binary digits are directly understood by the computer hardware, but they are difficult to deal with and comprehend by humans. This led to the invention of high-level languages which are close to natural languages and are easier to read, write, and maintain, but are not directly understood by the computer hardware. An advantage of using high-level languages is that they are portable, i.e., they can run on different types of computers with little or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten in order to run on another type of system. A wide variety of high-level languages, such as FORTRAN, C, C++, Java, Python, etc., exist.

IV. Coding

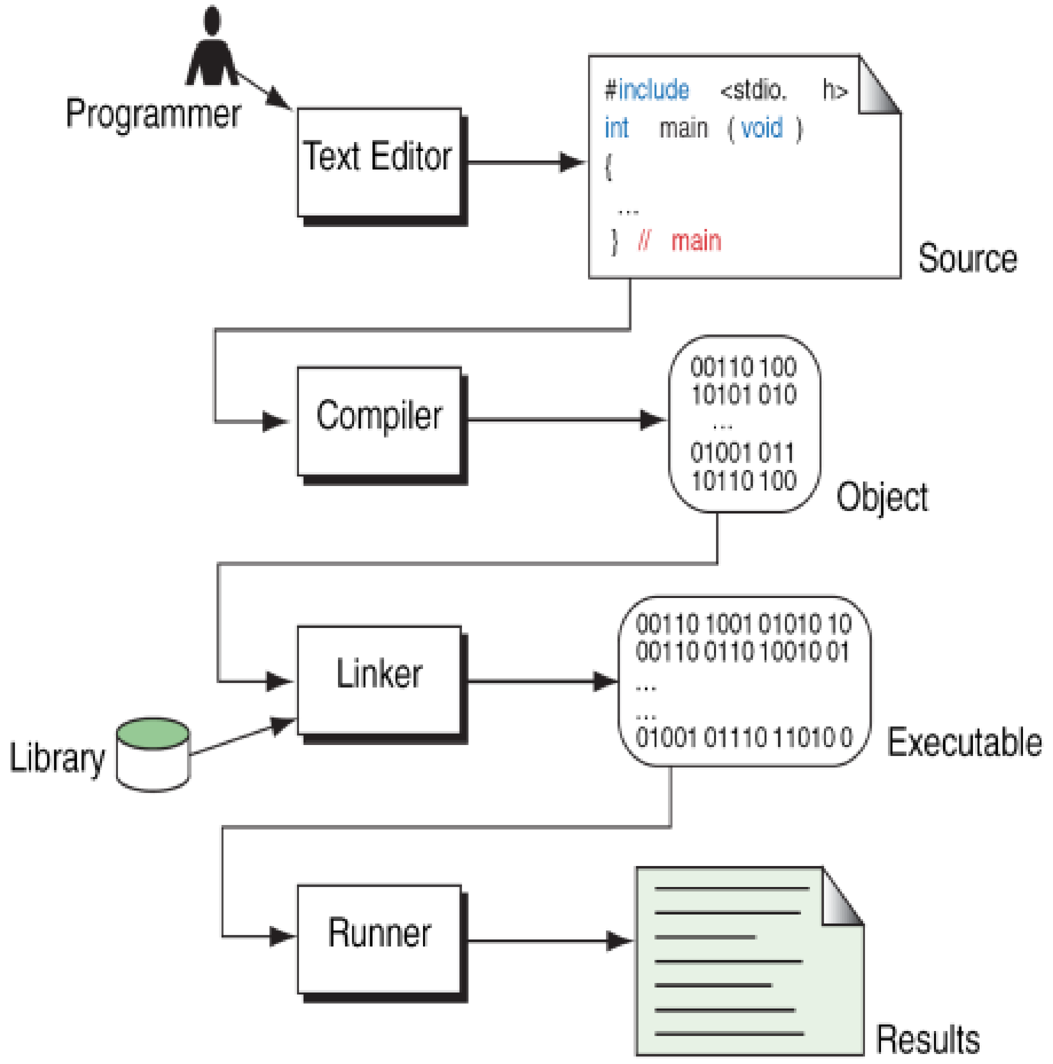
- ❑A program written in a high-level language is called source code. We need to translate the source code into machine language using a compiler or an interpreter, so that it can be understood by the computer.
- ❑ **Creating and Running Programs**

  There are four steps in this process.

  1. Writing and editing the program using Text editor (source code).
  2. Compile the program using any C compiler.(.bak file)
  3. Linking the program with the required library modules(object file)
  4. Executing the program. (.Exe file)

IV. Coding

❑ Creating and Editing a C Program in C Programming Language compiler:



What is C?

C is a programming language developed at **AT & T‟s Bell Laboratories of USA in 1972.** It was designed and written by **Dennis Ritche**.

IV. Coding

❑ General Structure of a C program:

A C program basically consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

```
#include <stdio.h>


int main()
{
   /* my first program in C */
   printf("Hello, World! \n");


   return 0;
}
```

IV. Coding

❑ General Structure of a C program:

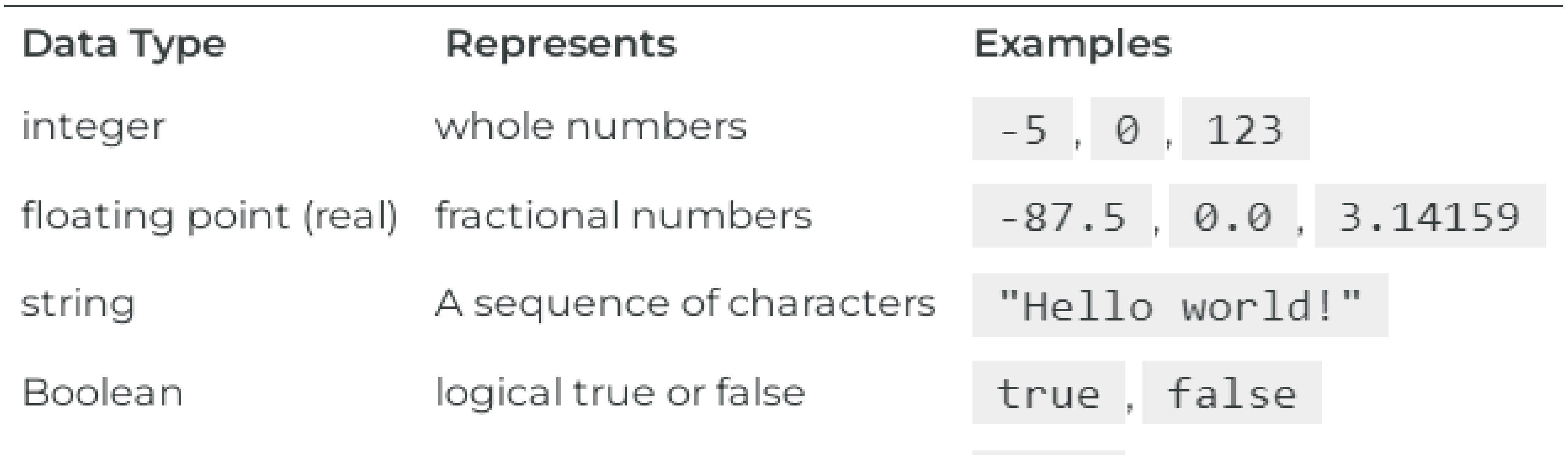
The various parts of the above program:

1. The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
2. The next line *int main()* is the main function where the program execution begins.
3. The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
4. The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
5. The next line **return 0;** terminates the main() function and returns the value 0.

v. Basic data types in C, Variables and Constants

❑Basic data types

A **data type** is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Most programming languages support various types of data, including integer, real, character or string, and Boolean.

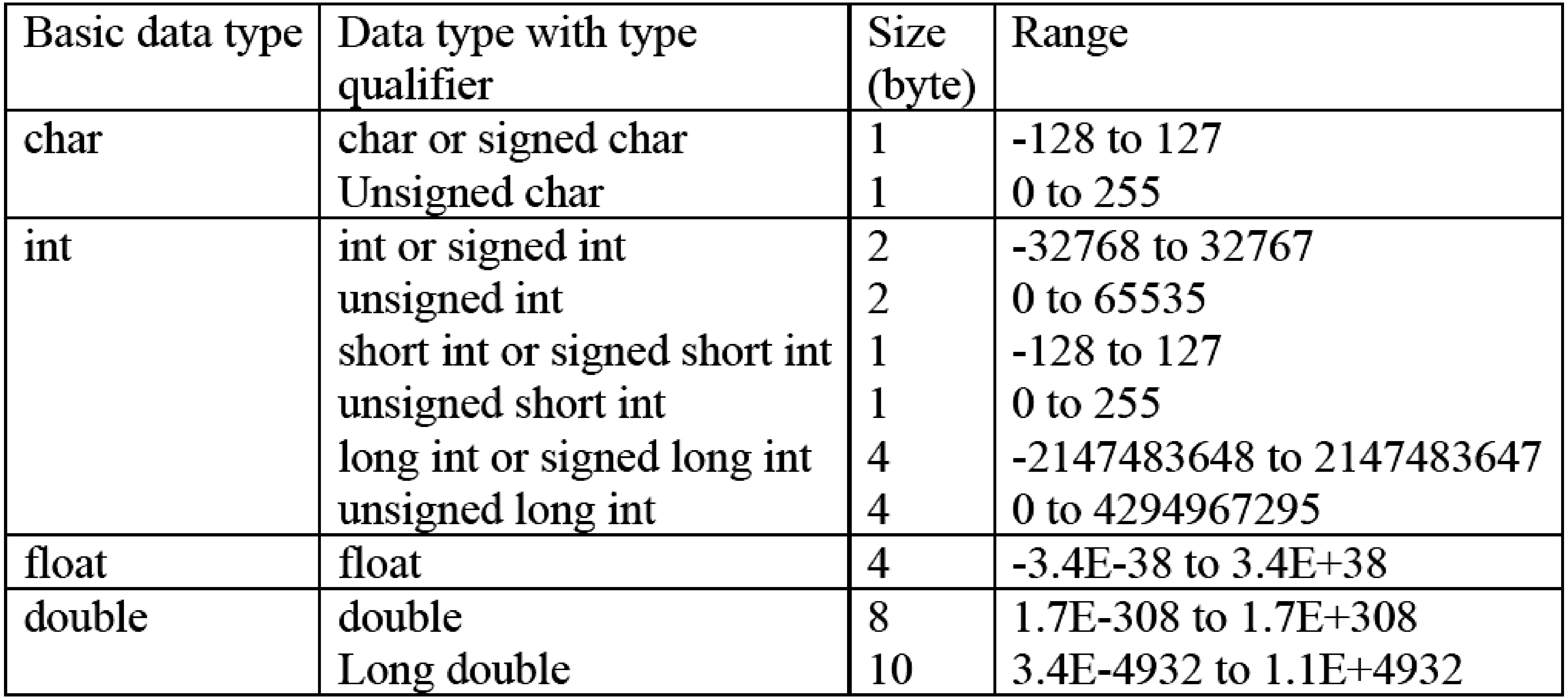


Data Type	Represents	Examples
integer	whole numbers	`-5`, `0`, `123`
floating point (real)	fractional numbers	`-87.5`, `0.0`, `3.14159`
string	A sequence of characters	`"Hello world!"`
Boolean	logical true or false	`true`, `false`

v. Basic data types in C, Variables and Constants

❑Basic data types

Size and Ranges of Data Types with Type Qualifiers



Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char	1	-128 to 127
	Unsigned char	1	0 to 255
int	int or signed int	2	-32768 to 32767
	unsigned int	2	0 to 65535
	short int or signed short int	1	-128 to 127
	unsigned short int	1	0 to 255
	long int or signed long int	4	-2147483648 to 2147483647
	unsigned long int	4	0 to 4294967295
float	float	4	-3.4E-38 to 3.4E+38
double	double	8	1.7E-308 to 1.7E+308
	Long double	10	3.4E-4932 to 1.1E+4932

v. Basic data types in C, Variables and Constants

❑ Variables

- ✓ A **variable** is a name of memory location. It is used to store data. Variables are changeable, we can change value of a variable during execution of a program. It can be reused many times.

- ✓ A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type.

v. Basic data types in C, Variables and Constants

❑ Variables

Rules to write variable names:

1. A variable name contains maximum of 30 characters/ Variable name must be upto 8 characters.
2. A variable name includes alphabets and numbers, but it must start with an alphabet.
3. It cannot accept any special characters, blank spaces except under score(_).
4. It should not be a reserved word.

Ex : i rank1 MAX min Student_name StudentName class_mark

v. Basic data types in C, Variables and Constants

❑ Variables

Declaration of Variables : A variable can be used to store a value of any data type. The declaration of variables must be done before they are used in the program. The general format for declaring a variable.

Syntax : data_type variable-1,variable-2,------, variable-n;

Variables are separated by commas and declaration statement ends with a semicolon.

Ex : int v;

```
int x,y,z;
float a,b;
char m,n;
```

v.Basic data types in C, Variables and Constants

❑ Variables

Assigning values to variables : values can be assigned to variables using the assignment operator (=). The general format statement is :

Syntax : variable = value;

Ex : x=100;

a= 12.25;

m=“f‟;

v.Basic data types in C, Variables and Constants

❑ Variables

Variable initialization:

When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as

Data type variable name = constant;

Example: int a=20;

Or int a;

a=20;

v. Basic data types in C, Variables and Constants

❑ Variables

Expressions

An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:

int z= x+y // arithmatic expression

a>b //relational

a==b // logical

cos(a) // function call

v. Basic data types in C, Variables and Constants

❑ Constants

Constants **refer** to fixed values that do not change during the execution of a program.

You can use **const** prefix to declare constants with a specific type as follows:

```
const type variable = value;
```

example:

```
const int i = 18;
const float Pi = 3,14;
```

VI.Operators: Arithmetic, Relational, Assignment And Logical

This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation.

Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, relational and others.

VI.Operators: Arithmetic, Relational, Assignment And Logical

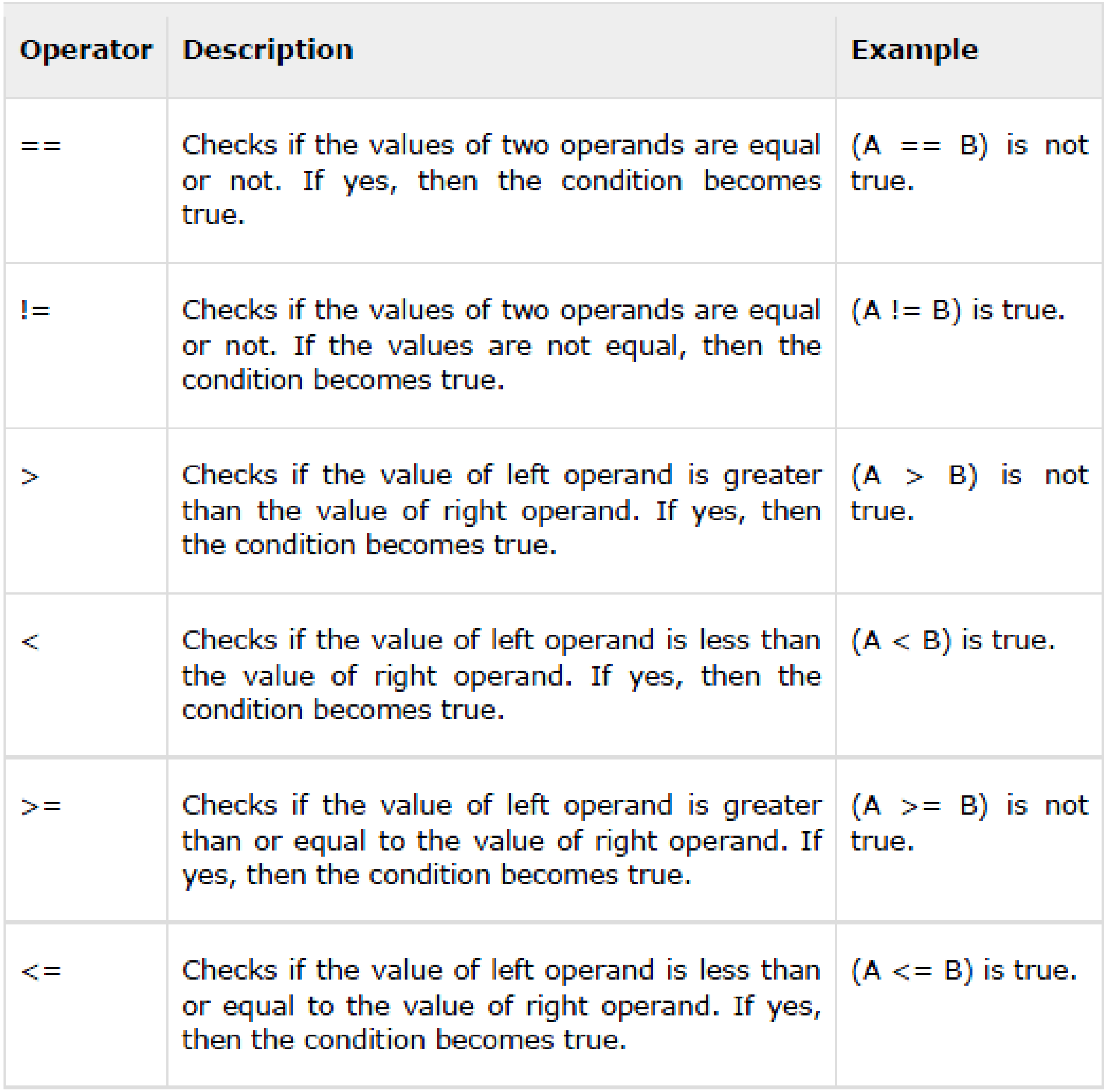
1. Arithmatic Operator

Operator	Description	Example
+	Adds two operands.	C= A+B;
-	Subtracts second operand from the first.	C= A-B;
*	Multiplies both operands.	C= A*B;
/	Divides numerator by de-numerator.	C= B/A;
%	Modulus Operator and remainder of after an integer division.	C= B%A;

VI.Operators: Arithmetic, Relational, Assignment And Logical

2. Relational Operators

int A=10, B=21;



Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

VI.Operators: Arithmetic, Relational, Assignment And Logical

3. Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression.

Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the compound statement. For example:

```
int x= y;
int Sum = x + y + z;
```

VI.Operators: Arithmetic, Relational, Assignment And Logical

4. Logical or Boolean Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression. C has three logical operators :

Operator	**Meaning**
&&	AND
||	OR
!	NOT

VII. Input / Output (I/O) Functions in C

In „C‟ language, two types of **Input/Output** functions are available, and all input and output operations are carried out through function calls. Several functions are available for input / output operations in „C‟. These functions are collectively known as the standard i/o library.

Input: In any programming language input means to feed some data into program. This can be given in the form of file or from command line.

Output: In any programming language output means to display some data on screen, printer or in any file.

VII. Input / Output (I/O) Functions in C

Printf and scanf

printf and **scanf** are two standard C programming language functions for input and output. Both are functions in the stdio library which means

```
#include <stdio.h>
```

is required at the top of your file.

VII. Input / Output (I/O) Functions in C

printf() function is used to print/display values of variables using the standared output device (monitor).

Syntax : printf(" format string " , variable_1,-------, variable_n);

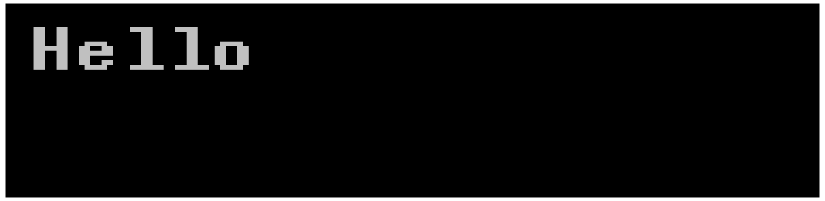
where variable_1, ,-------, variable_n are variables whose values are to be display in the monitor.

" format string " is the control string which represents the format specification.

VII. Input / Output (I/O) Functions in C

Example1: `printf("Hello");`

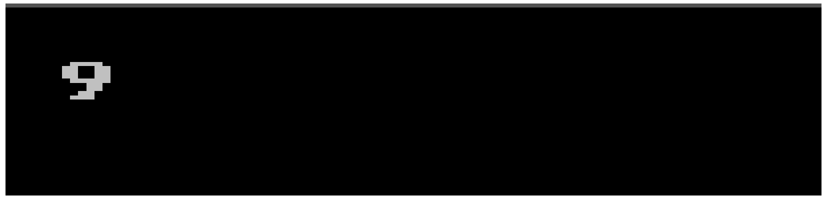
output →



```
Hello
```

Example2:
```
int number = 9;
printf("%d", number);
```

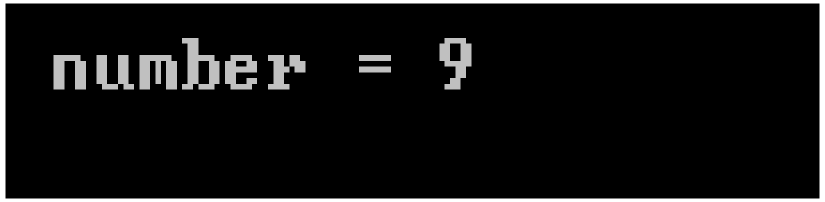
output →



```
9
```

Example3:
```
int number = 9;
printf(" number = %d", number);
```

output →



```
number = 9
```

VII. Input / Output (I/O) Functions in C

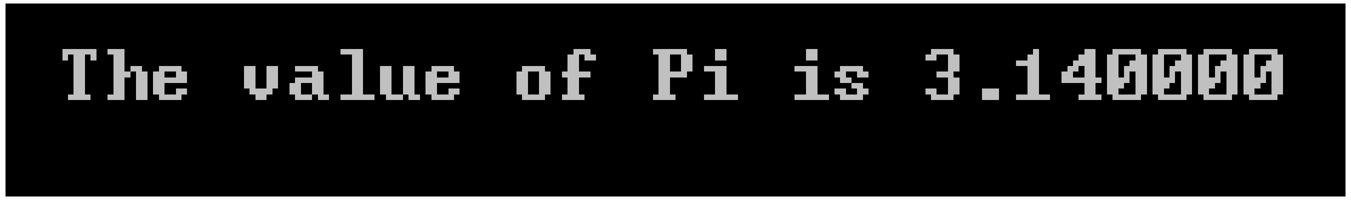
Example4: printf("%d");

output

Example5: float Pi = 3,14;

printf("The value of Pi is %f", Pi);

output

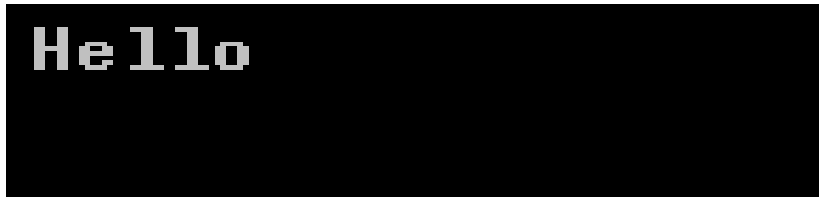


```
The value of Pi is 3.140000
```

Example6: char [255] word ="Hello";

printf("%s", word);

output



```
Hello
```

VII. Input / Output (I/O) Functions in C

Some commonly used format specifiers for qualified numbers with printf() are listed below:

PLACEHOLDER	DATATYPE
%d	integer
%u	unsigned int
%f	float
%lf	double floating-point
%c	char
%s	String

Note: you use the escape sequence "\n" to represent a new line or line break. It's a common way to indicate that text should continue on a new line.

VII. Input / Output (I/O) Functions in C

2. scanf() function is used to read/input values of variables using the standard input device (keyboard). It has the following form

Syntax :

scanf("control string ",&var_1, &var_2,----, &var_n);

where var_1, var_2, . . . , var_n are variables whose values are to be read from the keyboard.

"format string" is the control string which represents the format specification.

The symbol & (ampersand) represents the memory address where the variable value is to be stored.

VIII.Mathematical functions in C

In C, you can define and use mathematical functions by including the header:

#include <math.h>

which provides a wide range of mathematical functions.

Power and Square Root Functions:

pow(x, y): Calculates x raised to the power of y.

```
flaot result1;
result1 = pow(2, 3);    // result1 = 8.0
flaot result2;
result2 = sqrt(16); // result2 = 4.0
```

VIII.Mathematical functions in C

Trigonometric Functions:

- **sin(x)**: Calculates the sine of **x** (in radians).
- **cos(x)**: Calculates the cosine of **x** (in radians).
- **tan(x)**: Calculates the tangent of **x** (in radians).
- **asin(x)**: Calculates the arcsine (inverse sine) of **x**.
- **acos(x)**: Calculates the arccosine (inverse cosine) of **x**.
- **atan(x)**: Calculates the arctangent (inverse tangent) of **x**.

VIII.Mathematical functions in C

Exponential and Logarithmic Functions:

- **exp(x)**: Calculates the exponential function e^x.
- **log(x)**: Calculates the natural logarithm (base e) of **x**.
- **log10(x)**: Calculates the base-10 logarithm of **x**.

Absolute Value Functions:

- **fabs(x)**: Calculates the absolute value of **x**.

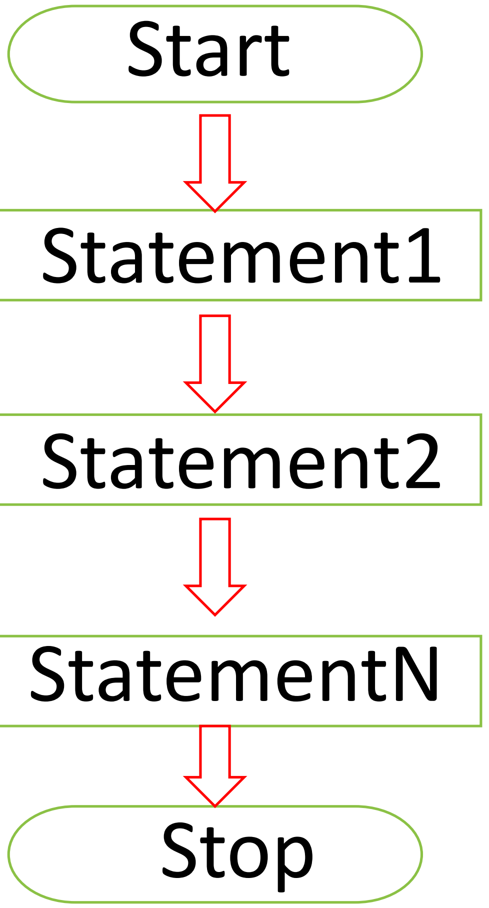
IX. Conditional Statements

Control statements control the flow of execution of the statements of a program. The various types of control statements in C language are as under:

1) Sequential actions.
2) Conditional actions.
3) Repeat actions.

IX. Conditional Statements

Sequential control: In sequential control, the C program statements are executed sequentially i.e., one after the another from beginning to end.



IX. Conditional Statements

Sequential control

Example :

```
#include<stdio.h>
int main()
{
float x , y, z;
x=3.5;
y=x+10;
y=y-x;
x=1;
z=y/x+2.5;
getch();
return 0;
}
```

Execution trace:

	x	y	z
Begin	?	?	?
1	3.5	?	?
2	3.5	13.5	?
3	3.5	10.0	?
4	1.0	10.0	?
5	1.0	10.0	12.5
end	1.0	10.0	12.5

IX. Conditional Statements

Conditional Control (Selection Control or Decision Control) : In conditional control , the execution of statements depends upon the condition-test. If the condition evaluates to true, then a set of statements is executed otherwise another set of statements is followed. This control is also called Decision Control because it helps in making decision about which set of statements is to be executed.

Decision control structure in C can be implemented by using:

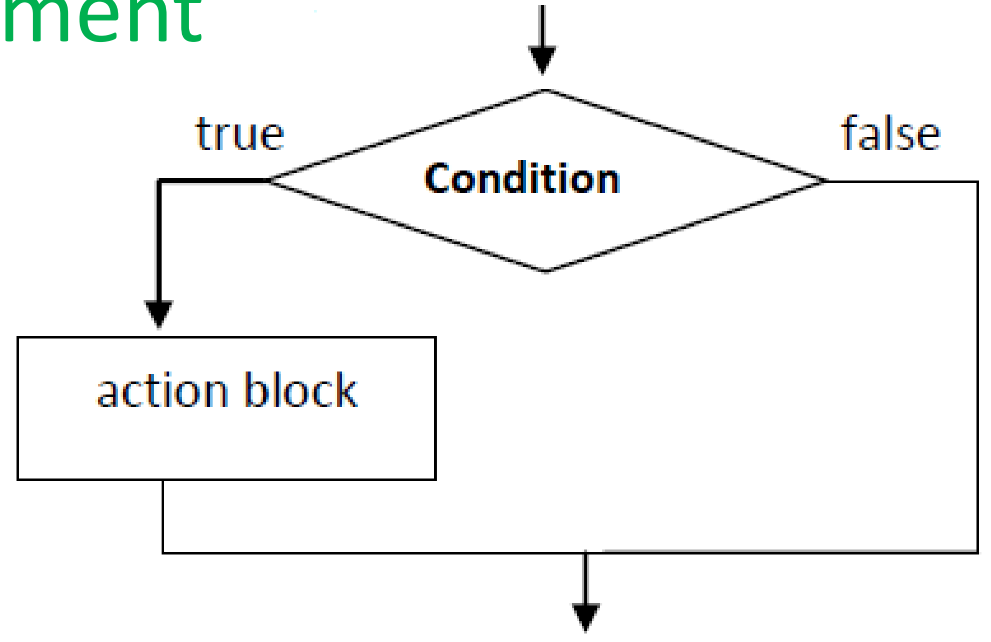
- if statement
- if-else statement
- Nested if else statement
- case control structure

IX. Conditional Statements

Simple conditional action: if statement

Syntax:

```
……….
if(condition)
{
<Action Block (statements) >
}
……….
```



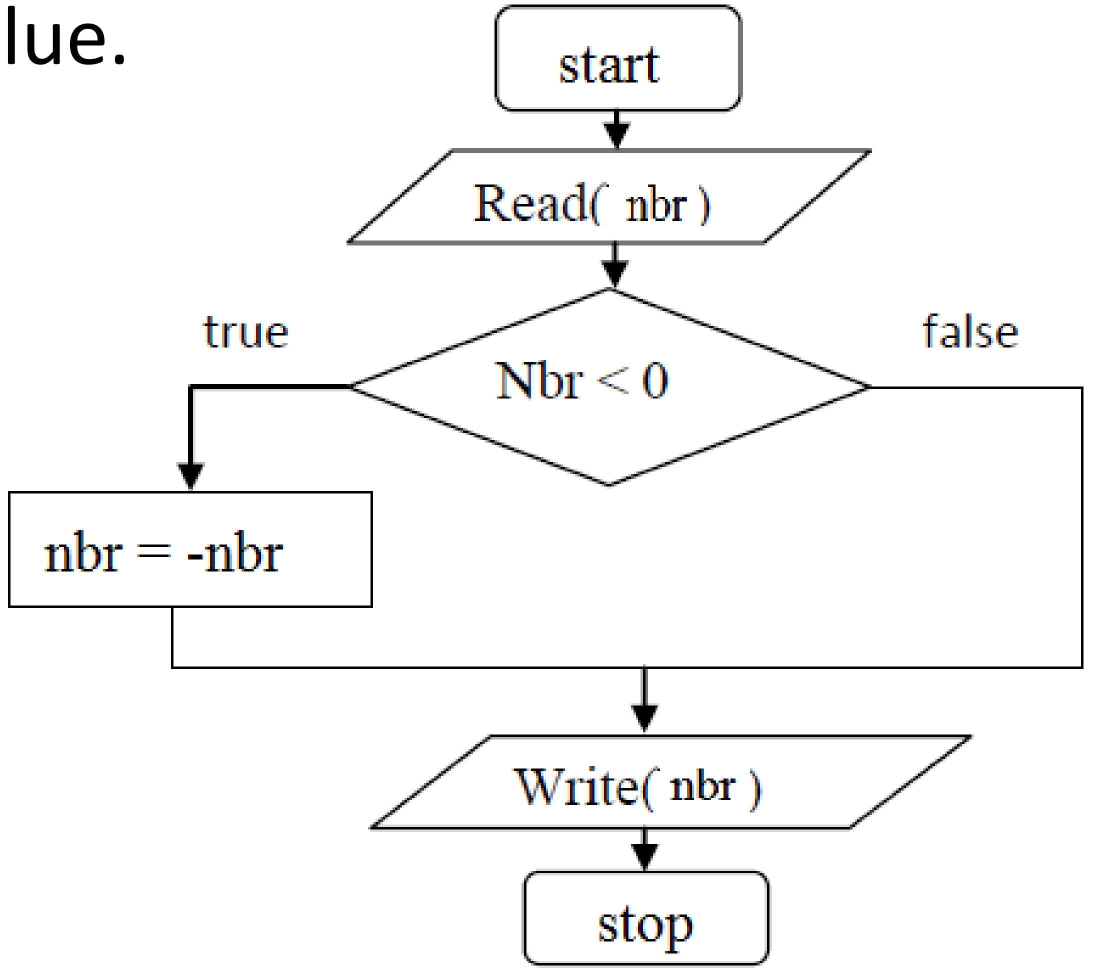
- ➢ It consists of two parts: condition and action.
- ✓ The (condition) part describes a state which can be true or false (Boolean type expression).
- ✓ The <Action Block> part represents a piece of an algorithm (one or more instructions).

IX. Conditional Statements

Simple conditional action: if statement

Example: Write a C program that reads a real number identified by "nbr", then gives its absolute value.

```
#include<stdio.h>
int main()
{
float nbr;
scanf("%f ",& nbr);
if (nbr<0)
{
nbr=-nbr;
}
printf(" %f", nbr);
getch();
return 0;
}
```

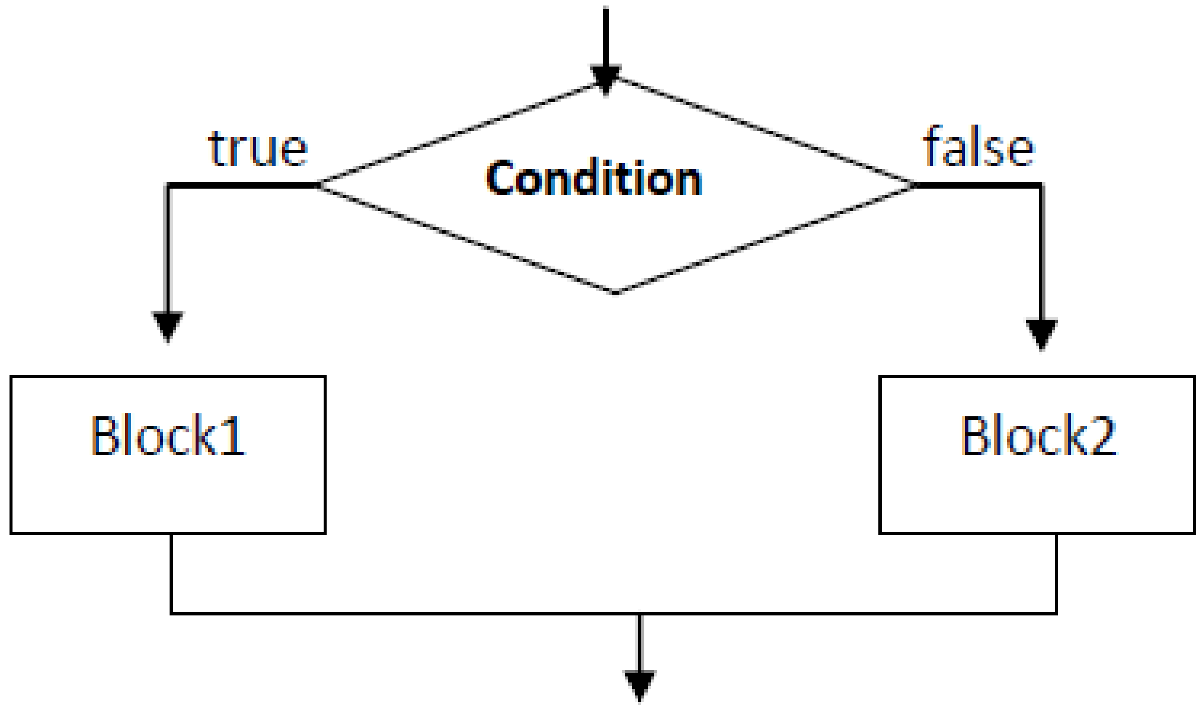


IX. Conditional Statements

Alternative action: if-else statement: This is a bi-directional control statement. This statement is used to test a condition and take one of the two possible actions. If the condition evaluates to true then one statement (or block of statements) is executed otherwise other statement (or block of statements) is executed.

Syntax:

```
………
If (condition)
{
< Block of (instructions) >
} actions1
else
{
< Block of actions2(instructions) >
 }
………
```



IX. Conditional Statements

if-else statement:

Example: Write an C program that reads two real numbers and then determines the biggest of them?

```
#include<stdio.h>
int main()
{
float x,y,Max;
scanf("%f ",&x);
scanf("%f ",&y);
if (x>y)
{
Max=x;
}
Else
{
Max=y;
}
printf(« The biggest number is: %f ", Max);
getch();
return 0;
}
```

IX. Conditional Statements

Nested if else statement:

If we have if-else statement within either the body of an if statement or the body of else statement or in the body of both if and else, then this is known as nesting of if else statement. The general form of nested if-else statement is as follows:

```
If (condition1)
{
        if (condition2)
        {
        statements;
         }
        else
        {
        statements;
        }
}
```

```
else
{
        if (condition3)
        {
        statements;
         }
        else
        {
        statements;
        }
}
```

IX. Conditional Statements

Nested if else statement:

Example: program to find the largest of three numbers

```
#include<stdio.h>
int main()
{
int a, b,c,large;
printf(“enter the three numbers”);
scanf(“%d%d%d”,&a,&b,&c);
if(a>b)
{
if(a>c)
large=a;
else
large=c;
}
else
{
if(b>c)
large=b;
else
large=c;
}
printf(“largest number is %d”,large);
getch();
return 0;
}
```

IX. Conditional Statements

Multiple choice action: switch case statement

This statement is used to select one out of the several numbers of alternatives present in a block. This selection statement 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.

IX. Conditional Statements

Multiple choice action: switch case statement

The general syntax of switch case statement is as follows:

```
switch(expression)
{
case constant1: statements;
case constant2: statements;
case constant3: statements;
………………………
………………………
case constantN: statements;
default : statement;
}
```

IX. Conditional Statements

Multiple choice action: switch case statement

Example: program to print day of the week

```
#include<stdio.h>
int main()
{
int day;
printf("enter the day number from 1 to 7 \n");
scanf("%d",&day);
switch(day)
{
case 1: printf("monday");
break;
case 2:printf("tuesday");
break;
case 3: printf("wednesday");
break;
case 4: printf("thursday");
break;
case 5: printf("friday");
break;
case 6: printf("saturday");
break;
case 7: printf("sunday");
break;
default :printf("wrong input");
}
getch();
return 0;
}
```

IX. Conditional Statements

Boolean Expression & Operators

- ❑ Boolean expressions are the expressions that evaluate a condition and result in a Boolean value i.e true or false.

- ❑ Boolean expressions are written using Boolean operators (AND) **&&**, (OR)**||** and (NOT) **!**.

IX. Conditional Statements

Boolean Expression & Operators

- (condition1 AND condition2)

condition1	condition2	condition1 && condition2
true	true	true
true	false	false
false	true	false
false	false	false

- (condition1 OR condition2)

condition1	condition2	condition1 || condition2
true	true	true
true	false	true
false	true	true
false	false	false

IX. Conditional Statements

Boolean Expression & Operators

- NOT (condition)

(condition)	! (condition)
true	false
false	true

Example:

```
float x,y,z;
x= 5;
y=2.1;
z=-4.5;
```

((x>y)&&(z==x)) $\longrightarrow$ false

((x>y)||(z<x)) $\longrightarrow$ true

(!(x==y)&&(z<x)) true

X. Iteration Statements (loops)

- The looping can be defined as repeating the same process multiple times until a specific condition satisfies. It is known as iteration also.
- A ***loop*** is a statement whose job is to repeatedly execute some other statement (the ***loop body***).
- The looping simplifies the complex problems into the easy ones.
- It enables to alter the flow of the program so that instead of writing the same code again and again, we can execute the same code for a finite number of times.

X. Iteration Statements (loops)

- **For example,** if we need to print “Hello” 10-times then, instead of using the *printf* statement 10 times, we can use *printf* once inside a loop which runs up to 10 iterations.
- There are three types of loops in C language those are given below:

- ✓ **while**
- ✓ **do while**
- ✓ **for**

X. Iteration Statements (loops)

Essential components of a loop

- ✓ Counter
- ✓ Initialisation of the counter with initial value
- ✓ Condition to check with the optimum value of the counter
- ✓ Statement(s) to be executed by iteration
- ✓ Increment/decrement

X. Iteration Statements (loops)

A. The while Statement

The ***while*** loop in c is to be used in the scenario where the block of statements is executed in the ***while*** loop until the condition specified in the *while* loop is satisfied. It is also called a pre-tested loop.

syntax:

```
initialisation;
while(condition)
{
block of statements to be executed ;
Increment ;
}
```

X. Iteration Statements (loops)

A. The while Statement

Example: Write a C-program to print 10 natural numbers

```
#include<stdio.h>
int main ()
{
int i=1;
while (i<=10)
{
printf(”%d\n”,i);
i=i+1;
}
}
```

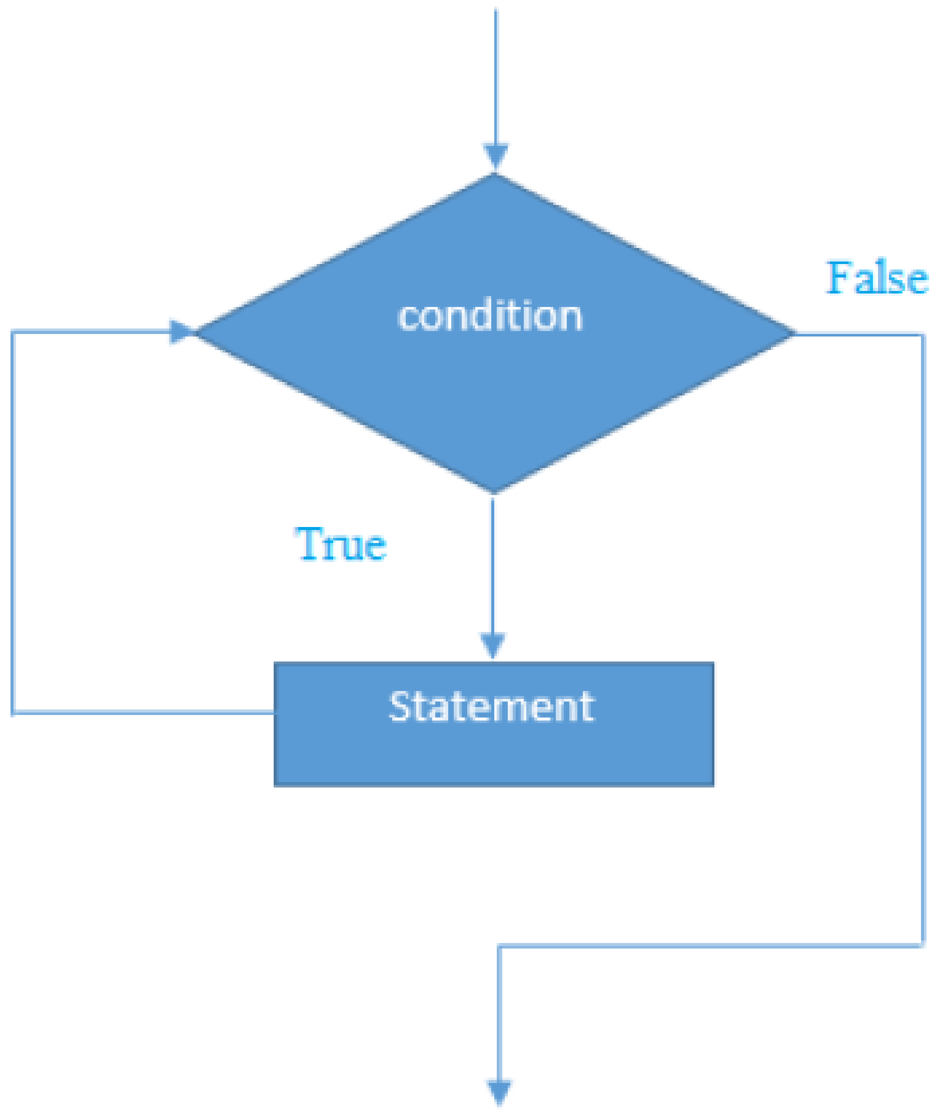
Output

```
1
2
3
5
6
7
8
9
10
```

X. Iteration Statements (loops)

A. The while Statement

Flowchart of while loop



X. Iteration Statements (loops)

B. The do-while Statement

The ***do-while*** loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

Syntax:

```
do
{
code to be executed ;
}
while(condition);
```

X. Iteration Statements (loops)

B. The do-while Statement

Example: Write a C-program to print 10 natural numbers

```
#include<stdio.h>
int main(){
int i=1;
do
{
printf("%d \n",i);
i++;
}
while(i<=10);
return 0;
}
```

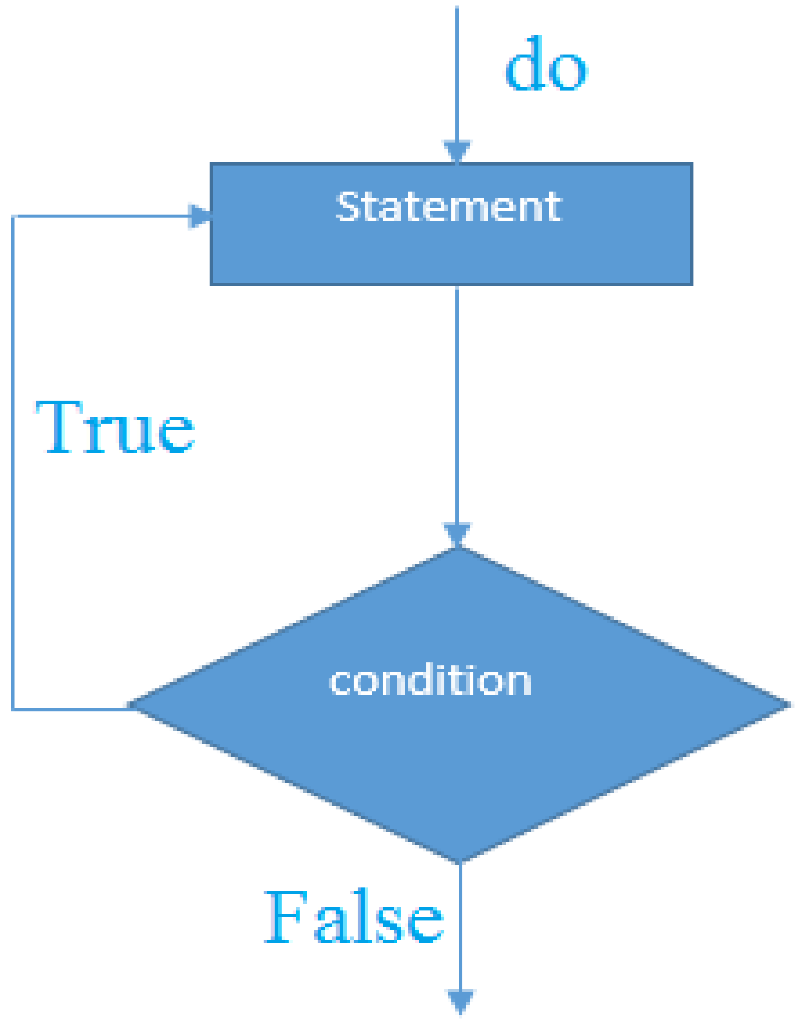
Output

```
1
2
3
5
6
7
8
9
10
```

X. Iteration Statements (loops)

B. The do-while Statement

Flowchart of do while loop



X. Iteration Statements (loops)

C. The for Statement

The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax:

```
for(Expression1; Expression2; Expression3)
{
codes to be executed;
}
```

X. Iteration Statements (loops)

C. The for Statement

Expression 1

- Represents the initialization of the loop variable.
- More than one variable can be initialised.

Expression 2

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false.

Other conditions will be treated as statements.

Expression 3

- Expression 3 is increment or decrement to update the value of the loop variable

X. Iteration Statements (loops)

C. The for Statement

Example: Write a C-program to print 10 natural numbers

```
#include<stdio.h>
int main(){
int i;
for(i=1;i<=10;i=i+1)
{
printf("%d\n", i);
}
return 0;
}
```

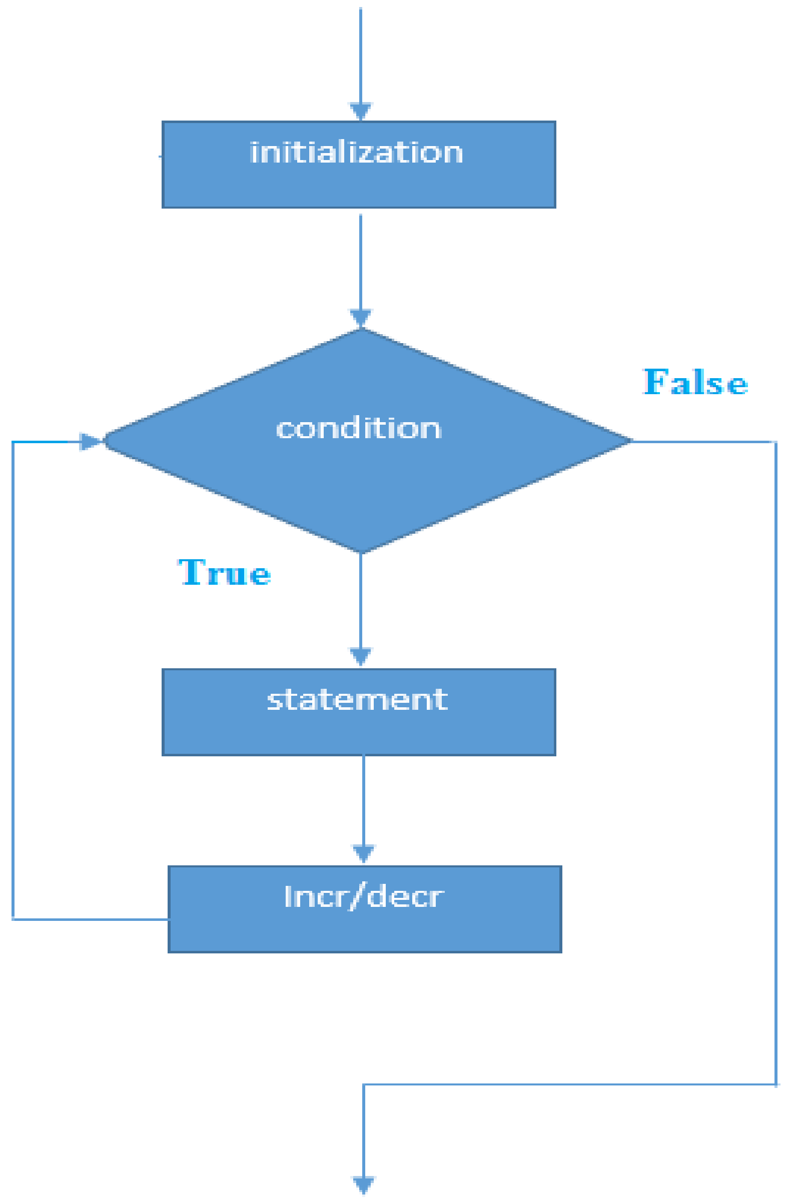
Output

```
1
2
3
5
6
7
8
9
10
```

X. Iteration Statements (loops)

C. The for Statement

Flowchart of for loop in C


initialization
condition
False
True
statement
Incr/decr