UNIT-1

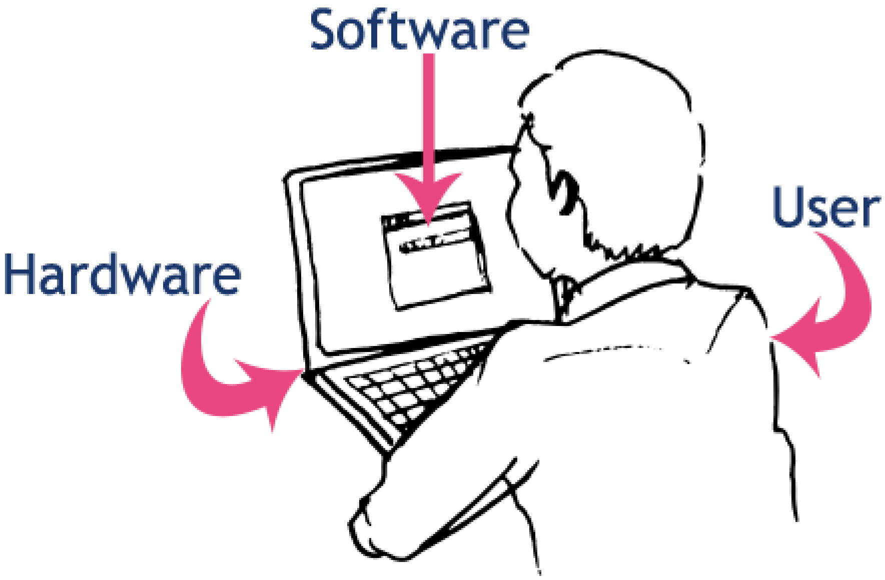
Computer Systems

Computer is an electronic device which operates under the control of instructions stored in its own memory. A computer can take data from the user through input devices (**Input**), process the user given data (**Processing**), produces the result to the user through output devices (**Output**)and stores data (**Information**) for future use. Computer can be defined as follows...

Computer is an electronic device which operates under the control of instructions stored in its own memory and it takes the data from the user, process that data, gives the result and stores the result for future use.

Every computer mainly consists of three things and those are...

1. Hardware
2. Software
3. User



Here the user interacts with the software, and the software makes the computer hardware parts to work for the user.

All physical components of the computer are called as computer hardware. A user can see, touch and feel every hardware of the computer. All hardware components perform any task based on the instructions given by the computer software.

The computer hardware is the physical part of a computer.

The computer hardware components are as follows...

1. **Input Devices** - These are the parts through which a user can give the data to the computer.
2. **Output Devices** - These are the physical components of a computer through which the computer gives the result to the user.
3. **Storage Devices** - These are the physical components of a computer in which the data can be stored.

Input Devices

Computer input devices are the physical components of the computer which are used to give the data given by the user to the computer. Using input devices the user can give the data to the computer.

Example

Keyboard

Mouse

Joy Stick

Mic

Barcode Reader

Stylus/Pen

Web Camera

Touch pad

Touch Screen

Finger Print reader

Output Devices

Computer output devices are the physical components of the computer which are used to give the computer result to the User. Using output devices, the user can see the computer generated result.

Example

Monitor

Printer

Speakers

Head Set

Projector

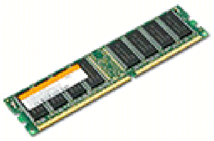
Plotter

Storage Devices

Computer storage devices are the physical components of the computer which are used to store data internally or externally.

Example

Hard Disk


RAM

ROM

CD/DVD

Floppy

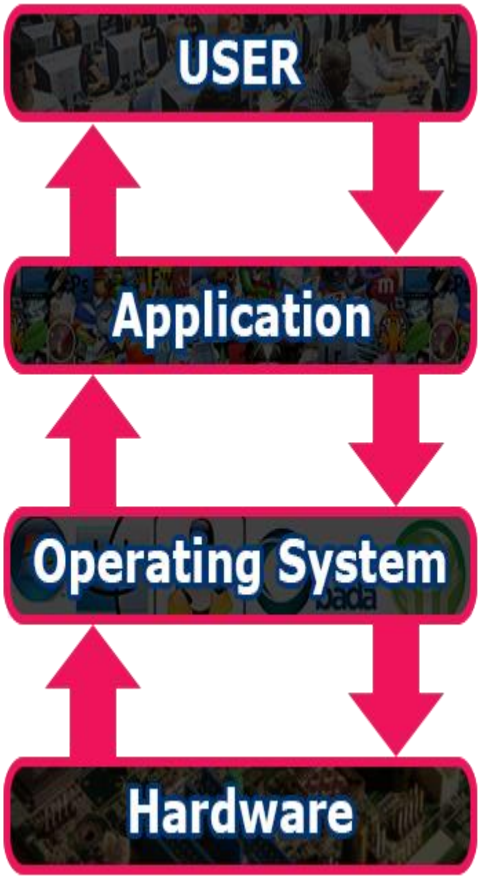
Memory Card

Pen Drive

Tape

When a user wants to communicate with the computer, the user interacts with an application. The application interacts with the operating system, and the operating

system makes hardware components to work according to the user given instructions. The hardware components sends the result back to the operating system, then the operating system forwards the same to the application and the application shows the result to the user.
By using input devices, the user interacts with the application and the application uses output devices to show the result. All input and output devices work according to the instructions given by the operating system.
The working process of a computer is shown in the following figure...



The Memory unit is classified into two types.
They are
1) Primary Memory
2) Secondary Memory

Primary memory: The following are the types of memoruies which are treated as primary
ROM: It represents Read Only Memory that stores data and instructions even when the computer is turned off. The Contents in the ROM can‘t be modified once if they are written . It is used to store the BIOS information.
RAM: It represents Random Access Memory that stores data and instructions when the computer is turned on. The contents in the RAM can be modified any no. of times by instructions. It is used to store the programs under execution.

Cache memory: It is used to store the data and instructions referred by processor.

Secondary Memory: The following are the different kinds of memories
Magnetic Storage: The Magnetic Storage devices store information that can be read, erased and rewritten a number of times.
Example: Floppy Disks, Hard Disks, Magnetic Tapes
Optical Storage: The optical storage devices that use laser beams to read and write stored data.
Example: CD(Compact Disk),DVD(Digital Versatile Disk)

COMPUTER SOFTWARE

Software of a computer system can be referred as anything which we can feel and see.
Example: Windows, icons
Computer software is divided in to two broad categories: system software and application software .System software manages the computer resources .It provides the interface between the hardware and the users. Application software, on the other hand is directly responsible for helping users solve their problems.

System Software

System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

The **operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

System support software provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.
The last system software category, **system development software**, includes the language translators that convert programs into machine language for execution ,debugging tools to ensure that the programs are error free and computer –assisted software engineering(CASE) systems.

Application software

Application software is broken in to two classes: general-purpose software and application –specific software. **General purpose software** is purchased from a software developer and can be used for more than one application. Examples of general purpose software include word processors, database management systems ,and computer aided design systems. They are labeled general purpose because they can solve a variety of user computing problems.

Application –specific software can be used only for its intended purpose.
A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed they cannot be used for other generalized tasks.
The relationship between system and application software is shown below. In this figure, each circle represents an interface point .The inner core is hard ware. The user is represented by the out layer. To work with the system, the typical user uses some form of application software. The application software in turn interacts with the operating system, which is a part of the system software layer. The system software provides the direct interaction with the hard ware. The opening at the bottom of the figure is the path followed by the user who interacts directly with the operating system when necessary.

Computer Languages

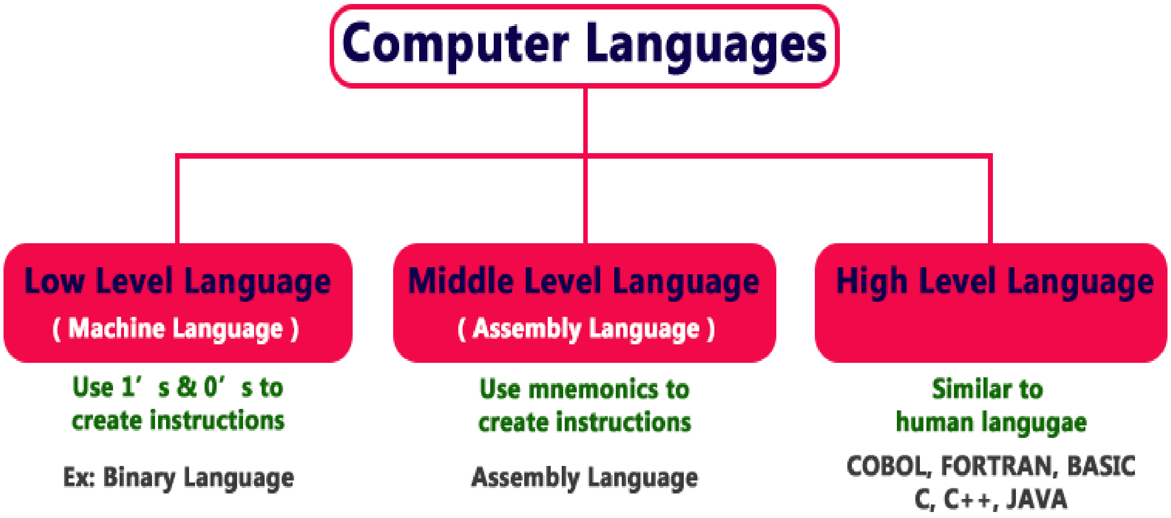
Generally, we use languages like english, hindi, telugu etc., to make communication between two persons. That means, when we want to make communication between two persons we need a language through which persons can express their feelings. Similarly, when we want to make communication between user and computer or between two or more computers we need a language through which user can give information to computer and vice versa. When user wants to give any instruction to the computer the user needs a specific language and that language is known as computer language.

User interacts with the computer using programs and that programs are created using computer programming languages like C, C++, Java etc.,

Computer languages are the languages through which user can communicate with the computer by writing program instructions.

Every computer programming language contains a set of predefined words and a set of rules (syntax) that are used to create instructions of a program.

Over the years, computer languages have been evolved from Low Level to High Level Languages. In the earliest days of computers, only Binary Language was used to write programs. The computer languages are classified as follows...



Low Level Language (Machine Language)

Low Level language is the only language which can be understood by the computer. **Binary Language** is an example of low level language. Low level language is also known as **Machine Language**. The binary language contains only two symbols 1 & 0. All the instructions of binary language are written in the form of binary numbers 1's

&0's. A computer can directly understand the binary language. Machine language is also known as **Machine Code**.

As the CPU directly understands the binary language instructions, it does not requires any translater. CPU directly starts executing the binary language instructions, and takes very less time to execute the instructions as it does not requires any translation. Low level language is considered as the First Generation Language (1GL).

Advantages

- A computer can easily understand the low level language.
- Low level language instructions are executed directly without any translation.
- Low level language instructions require very less time for thier execution.

Disadvantages

- Low level language instructions are very difficult to use and understand.
- Low level language instructions are machine dependent, that means a program written for a particular machine does not executes on other machine.
- In low level language, there is more chance for errors and it is very difficult to find errors, debug and modify.

Middle Level Language (Assembly Language)

Middle level language is a computer language in which the instructions are created using symbols such as letters, digits and special characters. Assembly language is an example of middle level language. In assembly language, we use predefined words called **mnemonics**. Binary code instructions in low level language are replaced with mnemonics and operands in middle level language. But computer can not understand mnemonics, so we use a translator called **Assembler** to translate mnemonics into binary language. Assembler is a translator which takes assembly code as input and produces machine code as output. That means, computer can not understand middle level language, so it needs to be translated into low level language to make it understandable by the computer. Assembler is used to translate middle level language to low level language.

Advantages

- Writing instructions in middle level language is easier than writing instructions in low level language.
- Middle level language is more readable compared to low level language.
- Easy to understand, find errors and modify.

Disadvantages

- Middle level language is specific to a particular machine architecture, that means it is machine dependent.
- Middle level language needs to be translated into low level language.
- Middle level language executes slower compared to low level language.

High Level Language

High level language is a computer language which can be understood by the users. High level language is very similar to the human languages and have a set of grammar rules that are used to make instructions more easily. Every high level language have a set of predefined words known as Keywords and a set of rules known as Syntax to create instructions. High level language is more easier to understand for the users but the computer can not understand it. High level language needs to be converted into low level language to make it understandable by the computer. We use **Compiler** or **interpretor** to convert high level language to low level language.

Languages like COBOL, FORTRAN, BASIC, C ,C++, JAVA etc., are the examples of high level languages. All these programming languages use human understandable language like english to write program instructions. These instructions are converted to low level language by the compiler so that it can be understood by the computer.

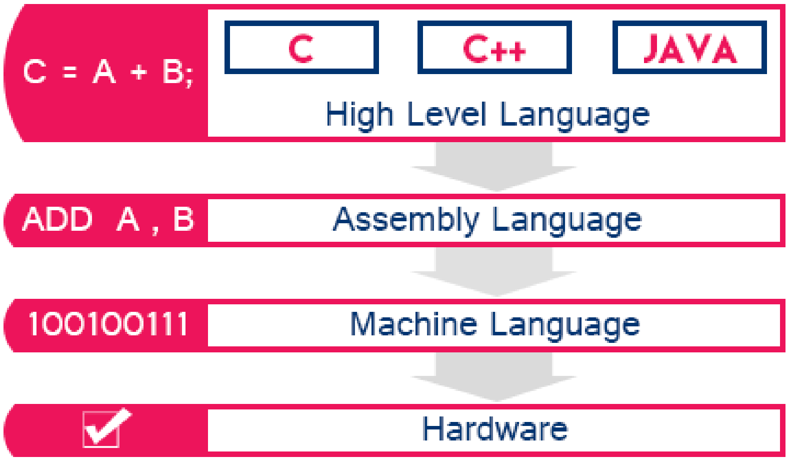
Advantages

- Writing instructions in high level language is more easier.
- High level language is more readable and understandable.
- The programs created using high level language runs on different machines with little change or no change.
- Easy to understand, create programs, find errors and modify.

Disadvantages

- High level language needs to be translated to low level language.
- High level language executes slower compared to middle and low level languages.

The follwing figure provides few key points related to the computer languages.



Language Translators
These are the programs which are used for converting the programs in one language into machine language instructions, so that they can be excuted by the computer.
1) Compiler: It is a program which is used to convert the high level language programs into machine language

2) Assembler: It is a program which is used to convert the assembly level language programs into machine language

3) Interpreter: It is a program, it takes one statement of a high level language program, translates it into machine language instruction and then immediately executes the resulting machine language instruction and so on.

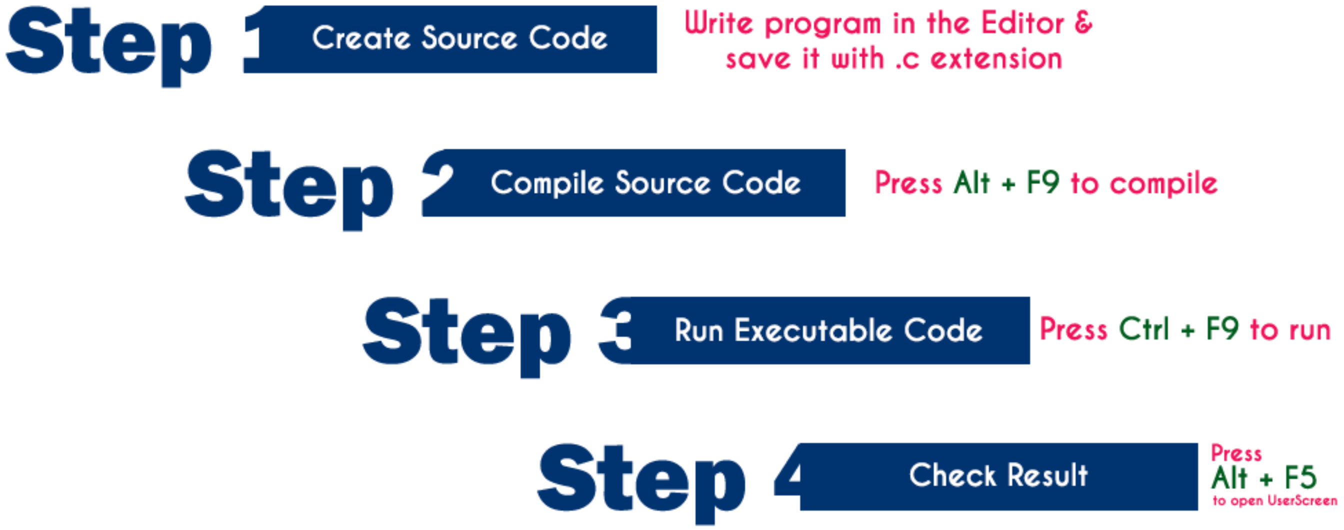
Creating and Running C Program

Generally, the programs created using programming languages like C, C++, Java etc., are written using high level language like English. But, computer cannot understand the high level language. It can understand only low level language. So, the program written in high level language needs to be converted into low level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.

Popular programming languages like C, C++, Java etc., use compiler to convert high level language instructions into low level language instructions. Compiler is a program that converts high level language instructions into low level language instructions. Generally, compiler performs two things, first it verifies the program errors, if errors are found, it returns list of errors otherwise it converts the complete code into low level language.

To create and execute C programs in Windows Operating System, we need to install

Turbo C software. We use the following steps to create and execute C programs in Windows OS…



Step 1: Creating Source Code

Source code is a file with C programming instructions in high level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C programming language rules. The following steps are used to create source code file in Windows OS…

- Click on **Start** button
- Select **Run**
- Type **cmd** and press Enter
- Type **cd c:\TC\bin** in the command prompt and press **Enter**
- Type **TC** press **Enter**
- Click on **File -> New** in C Editor window
- Type the **program**
- Save it as **FileName.c** (Use shortcut key **F2** to save)

Step 2: Compile Source Code (Alt + F9)

Compilation is the process of converting high level language instructions into low level language instructions. We use the shortcut key **Alt + F9** to compile a C program in **Turbo C**.

Compilation is the process of converting high level language instructions into low level language instructions.

Whenever we press **Alt + F9**, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any

Errors then compiler returns List of Errors, if there are no errors then the source code is converted into **object code** and stores it as file with **.obj** extension. Then the object code is given to the **Linker**. The Linker combines both the **object code** and specified **header file** code and generates an **Executable file** with **.exe** extension.

Step 3: Executing / Running Executable File (Ctrl + F9)

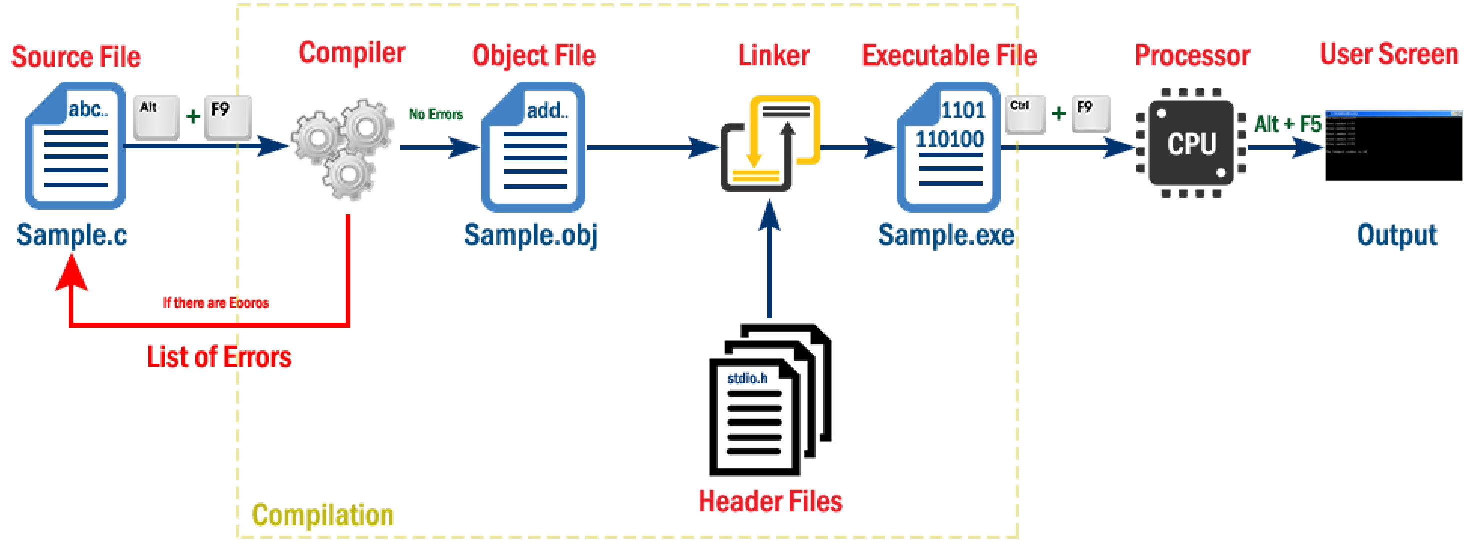
After completing compilation successfully, an executable file is created with **.exe** extension. The processor can understand this **.exe** file content so that it can perform the task specified in the source file.

We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe**file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called **User Screen**.

Step 4: Check Result (Alt + F5)

After running the program, the result is placed into **User Screen**. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key **Alt + F5** to open the User Screen and check the result.

When we execute a C program it undergoes with following process…



The file which contains c program instructions in high level language is said to be source code. Every c program source file is saved with .c extension, for example Sample.c.

Whenever we press Alt + F9 the source file is submitted to the compiler. Compiler checks for the errors, if there are any errors, it returns list of errors, otherwise generates object code in a file with name Sample.obj and submit it to the linker. Linker combines the code from specified header file into object file and generates executable file as Sample.exe. With this compilation process completes.

Now, we need to Run the executable file (Sample.exe). To run a program we press Ctrl + F9. When we press Ctrl + F9 the executable file is submitted to the CPU. Then CPU performs the task according to the instructions written in that program and place the result into UserScreen.

Then we press Alt + F5 to open UserScreen and check the result of the program.

ALGORITHM

Algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. No matter what the input values may be, an algorithm terminates after executing a finite number of instructions.
We represent an algorithm using a pseudo language that is a combination of the constructs of a programming language together with informal English statements.
The ordered set of instructions required to solve a problem is known as an *algorithm*.
The characteristics of a good algorithm are:

- Precision – the steps are precisely stated (defined).
- Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness – the algorithm stops after a finite number of instructions are executed.
- Input – the algorithm receives input.
- Output – the algorithm produces output.
- Generality – the algorithm applies to a set of inputs.

Example

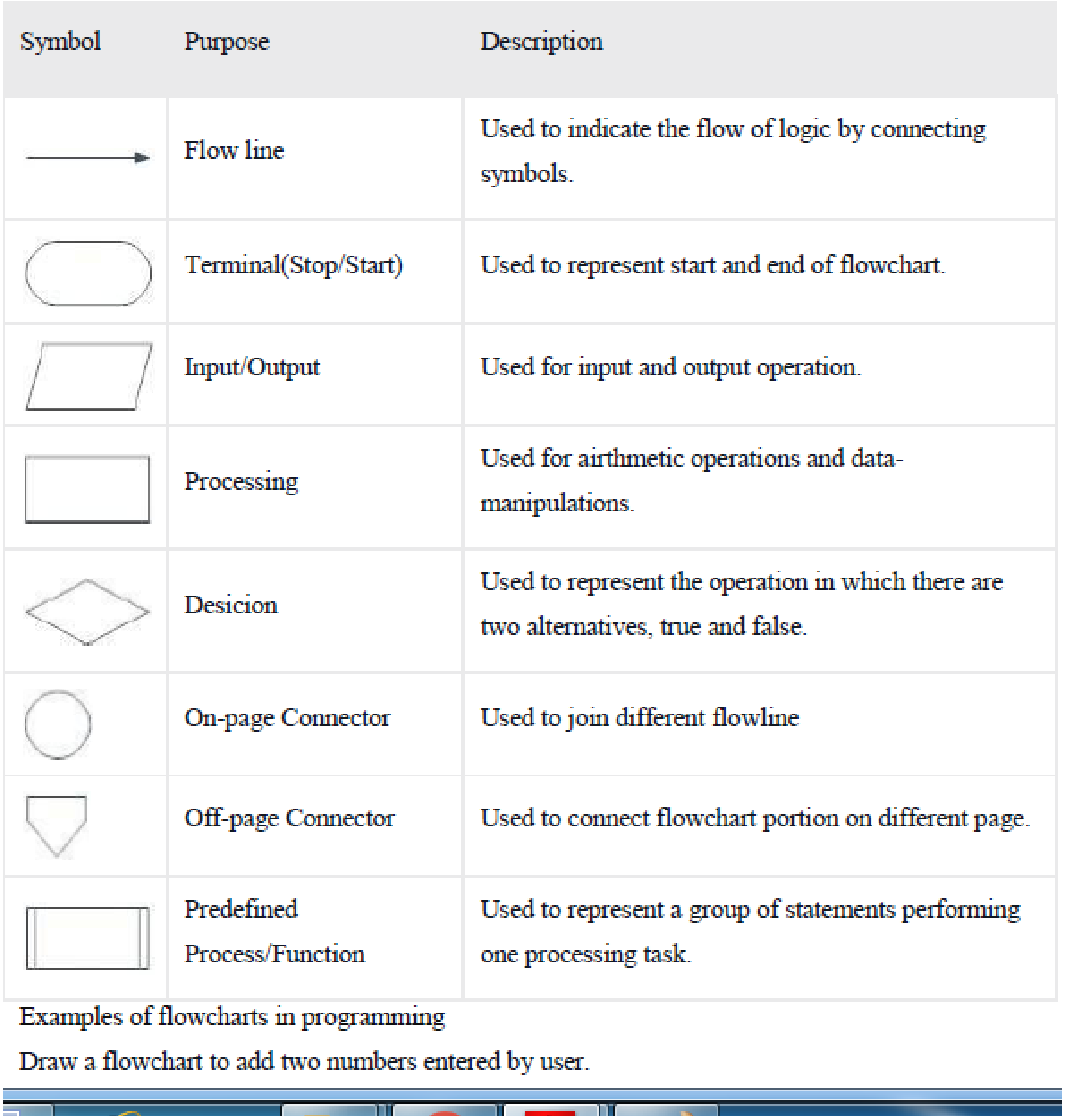
Q. Write a algorithem to find out number is odd or even?
Ans.

```
step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
print "number even"
else
print "number odd"
endif
step 5 : stop
```

FLOWCHART

Flowchart is a diagrammatic representation of an algorithm. Flowchart is very helpful in writing program and explaining program to others.

Symbols Used In Flowchart
Different symbols are used for different states in flowchart, For example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

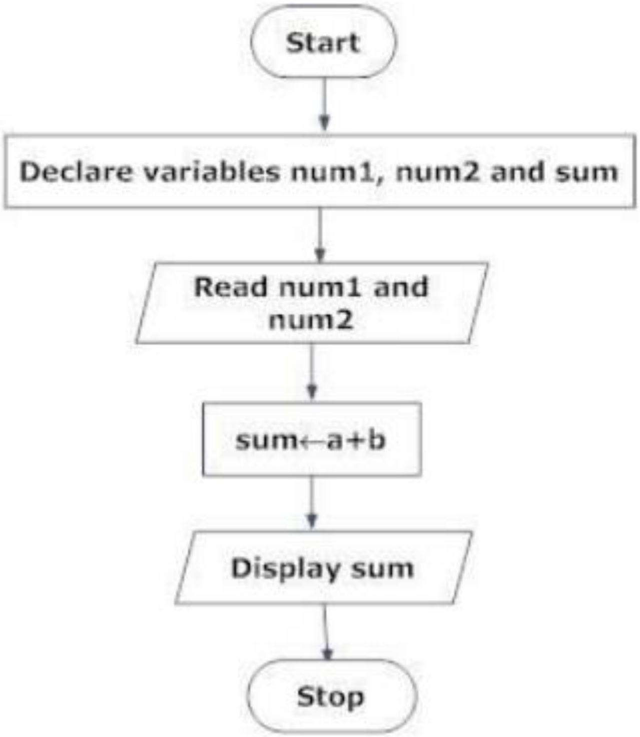


Symbol	Purpose	Description
→	Flow line	Used to indicate the flow of logic by connecting symbols.
⬭	Terminal(Stop/Start)	Used to represent start and end of flowchart.
▱	Input/Output	Used for input and output operation.
▭	Processing	Used for airthmetic operations and data-manipulations.
◇	Desicion	Used to represent the operation in which there are two alternatives, true and false.
○	On-page Connector	Used to join different flowline
⬠	Off-page Connector	Used to connect flowchart portion on different page.
◫	Predefined Process/Function	Used to represent a group of statements performing one processing task.

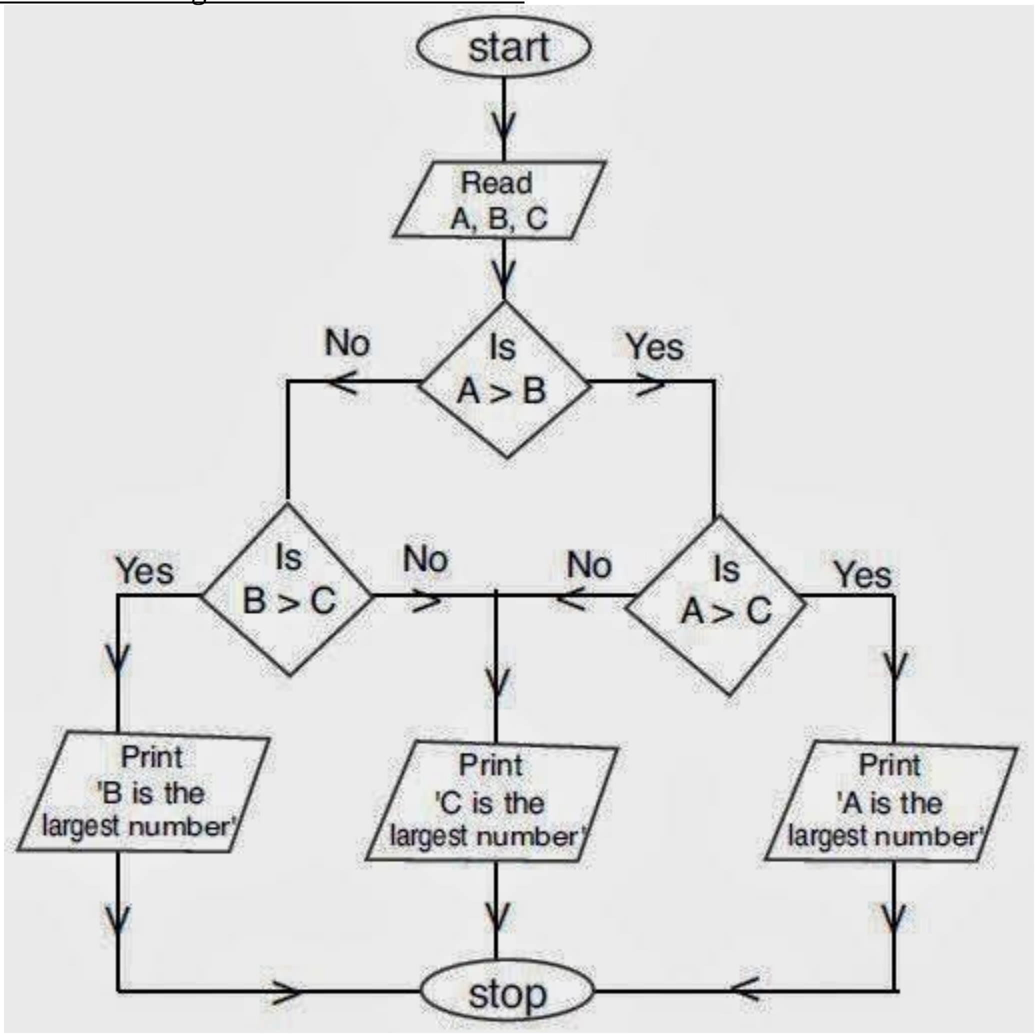
Examples of flowcharts in programming

Draw a flowchart to add two numbers entered by user.

Draw a flowchart to add two numbers entered by user.



Flow chart for greatest of three numbers:


start
Read A, B, C
Is A > B
No
Yes
Is B > C
Yes
No
No
Is A > C
Yes
Print 'B is the largest number'
Print 'C is the largest number'
Print 'A is the largest number'
stop

History of C Language

History of C language is interesting to know. Here we are going to discuss brief history of c language.

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Dennis Ritchie is known as the **founder of c language**.

It was developed to overcome the problems of previous languages such as B, BCPL etc.

Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.

Let's see the programming languages that were developed before C language.

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

Features of C Language

C is the widely used language. It provides a lot of **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed

8. Pointers
9. Recursion
10. Extensible

1) Simple

C is a simple language in the sense that it provides **structured approach** (to break the problem into parts), **rich set of library functions**, **data types** etc.

2) Machine Independent or Portable

Unlike assembly language, c programs **can be executed in many machines** with little bit or no change. But it is not platform-independent.

3) Mid-level prorgramming language

C is **also used to do low level programming**. It is used to develop system applications such as kernel, driver etc. It **also supports the feature of high level language**. That is why it is known as mid-level language.

4) Structured prorgramming language

C is a structured programming language in the sense that **we can break the program into parts using functions**. So, it is easy to understand and modify.

5) Rich Library

C **provides a lot of inbuilt functions** that makes the development fast.

6) Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7) Speed

The compilation and execution time of C language is fast.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array** etc.

9) Recursion

In c, we **can call the function within the function**. It provides code reusability for every function.

10) Extensible

C language is extensible because it **can easily adopt new features**.

C Tokens

Every C program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c program is called token. Every instruction in a c program is a collection of tokens. Tokens are used to construct c programs and they are said to the basic building blocks of a c program.

In a c program tokens may contain the following...

1. Keywords
2. Identifiers
3. Operators
4. Special Symbols
5. Constants
6. Strings
7. Data Types

Keywords in C

> A keyword is a **reserved word**. You cannot use it as a variable name, constant name etc. There are only 32 reserved words (keywords) in C language.

A list of 32 keywords in c language is given below:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

C Identifiers

In C programming language, programmers can specify their own name to variable, array, pointer, function, lable etc... Identifier is a collection of characters which acts as name of variable, function, array, pointer, strcture, lable etc... In other words, an identifier can be defined as user defined name to identify an enity uniquely in c programming language that name may be of variable name, function name, array name, pointer name, structure name or a lable.

Identifier is a user defined name of an entity to identify it uniquely during the program execution

Example

intmarks;
char studentName[30];
Here, **marks** and **studentName** are identifiers.

Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

Escape Sequence in C

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

List of Escape Sequences in C

Escape Sequence	Meaning

\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators

- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

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.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then –

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.

!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists	(A & B) = 12, i.e.,

	in both operands.	0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left	C += A is equivalent

	operand.	to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C

		= C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

There are 4 types of data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

Let's see the basic data types. Its size is given **according to 32 bit architecture**.

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767

signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

Variables in C

A **variable** is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring variable is given below:

```
int a;
float b;
char c;
```

Here, a, b, c are variables and int,float,char are data types.

We can also provide values while declaring the variables as given below:

```
int a=10,b=20;//declaring 2 variable of integer type
float f=20.8;
char c='A';
```

Rules for defining variables

- A variable can have alphabets, digits and underscore.
- A variable name can start with alphabet and underscore only. It can't start with digit.
- No white space is allowed within variable name.
- A variable name must not be any reserved word or keyword e.g. int, float etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

Inalid variable names:

```
int 2;
int a b;
int long;
```

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in C language.

1. Single Line Comments
2. Multi Line Comments

Single Line Comments

Single line comments are represented by double slash //. Let's see an example of single line comment in C.

```
#include<stdio.h>
void main(){
   //printing information
   printf("Hello C");
getch();;
}
```

Output:

```
Hello C
```

Even you can place comment after statement. For example:

```
printf("Hello C");//printing information
```

Mult Line Comments

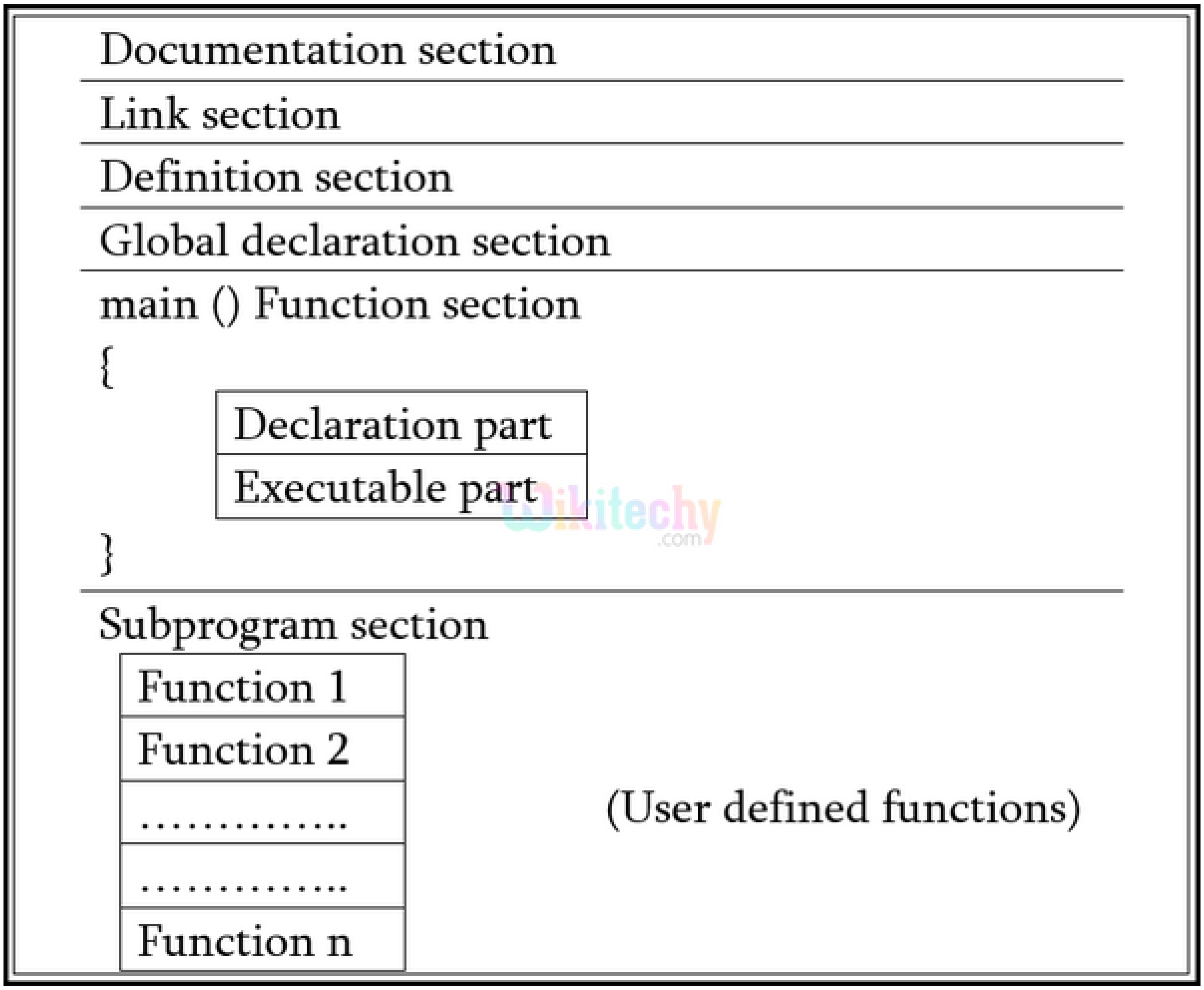
Multi line comments are represented by slash asterisk * ... *\. It can occupy many lines of code but it can't be nested. Syntax:

```
/*
code
to be commented
*/
```

Let's see an example of multi line comment in C.

```
#include<stdio.h>
void main(){
   /*printing information
     Multi Line Comment*/
   printf("Hello C");
getch();;
}
```

Basic Structure of C- Programming :



Input / Output Functions:

printf() and scanf() in C

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function

The **printf() function** is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

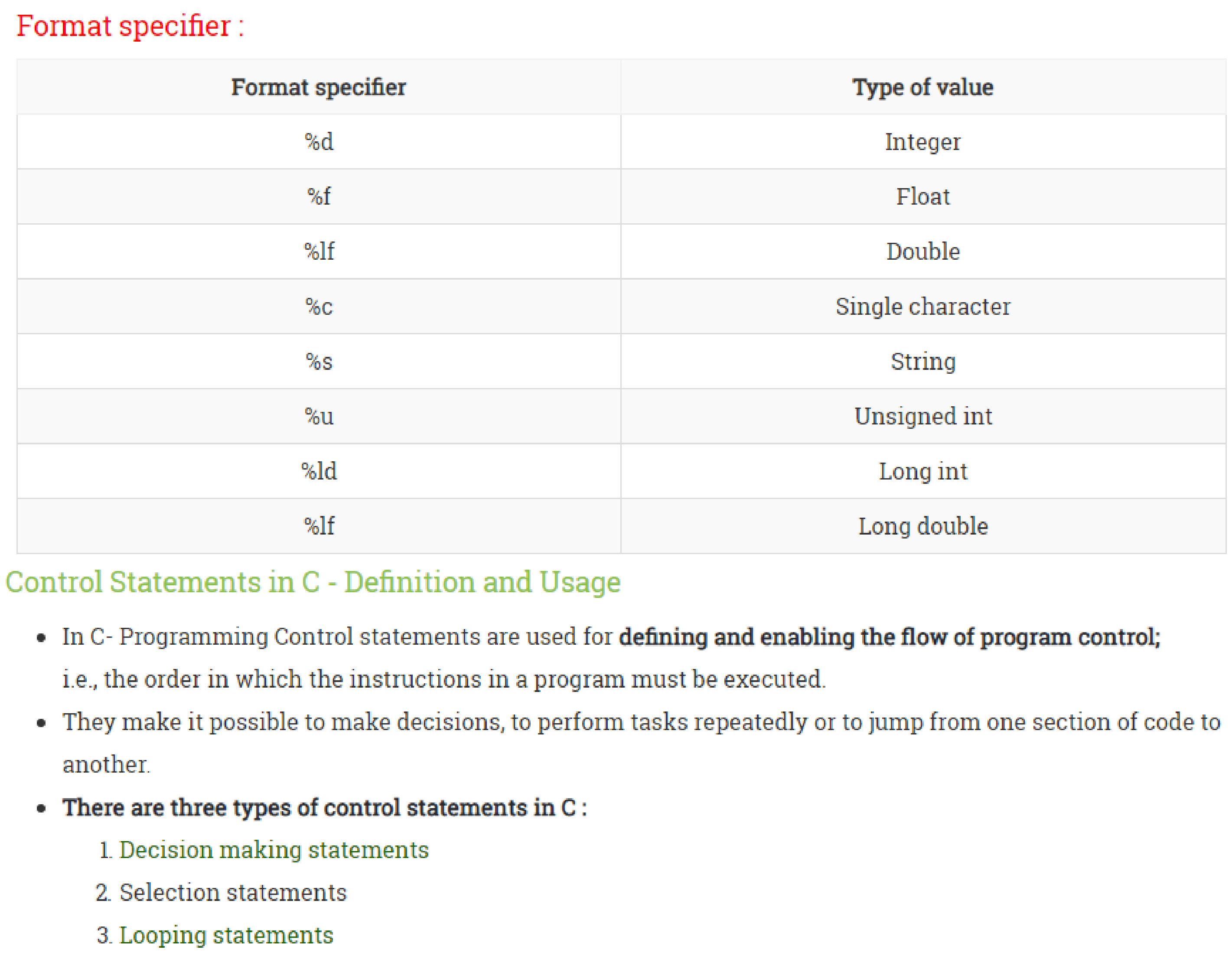
1. printf("format string",argument_list);

The **format string** can be %d (integer), %c (character), %s (string), %f (float) etc.

scanf() function

The **scanf() function** is used for input. It reads the input data from the console.

1. scanf("format string",argument_list);



Format specifier :

Format specifier	Type of value
%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	Long double

Control Statements in C - Definition and Usage

- In C- Programming Control statements are used for **defining and enabling the flow of program control;** i.e., the order in which the instructions in a program must be executed.
- They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.
- **There are three types of control statements in C :**
  1. Decision making statements
  2. Selection statements
  3. Looping statements

C if else Statement

The if statement in C language is used to perform operation on the basis of condition. By using if-else statement, you can perform operation either condition is true or false.

There are many ways to use if statement in C language:

- If statement
- If-else statement
- If else-if ladder
- Nested if

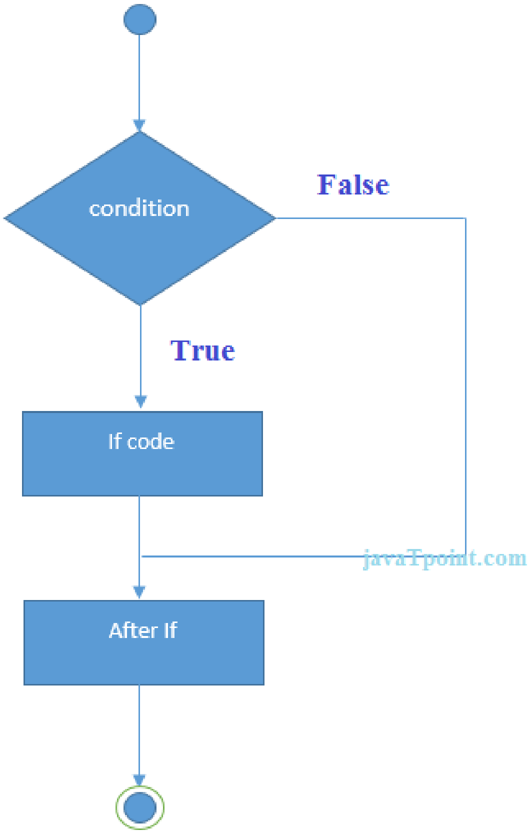
If Statement

It is one of the powerful conditional statement. If statement is responsible for modifying the flow of execution of a program. If statement is always used with a condition. The condition is evaluated first before executing any statement inside the body of If. The syntax for if statement is as follows:

The single if statement in C language is used to execute the code if condition is true. The syntax of if statement is given below:

```
if(expression){
//code to be executed
}
```

Flowchart of if statement in C



Let's see a simple example of c language if statement.

```
#include<stdio.h>
void main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
getch();;
}
```

Output

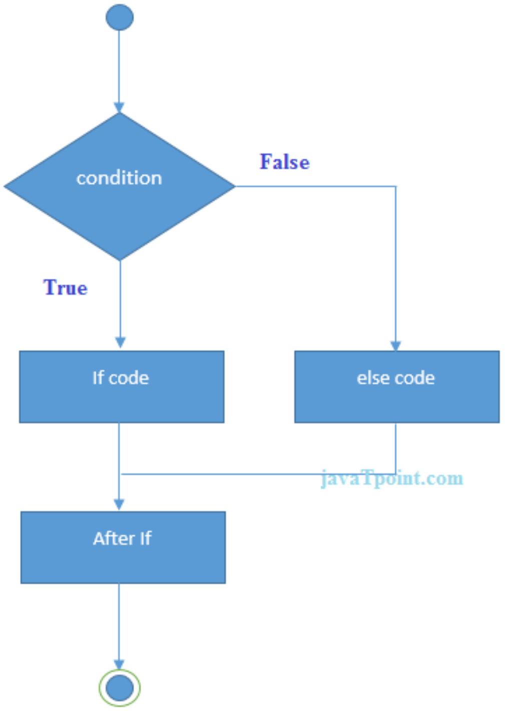
```
enter a number:4
4 is even number
enter a number:5
```

If-else Statement

The if-else statement in C language is used to execute the code if condition is true or false. The syntax of if-else statement is given below:

```
if(expression){
//code to be executed if condition is true
}else{
//code to be executed if condition is false
}
```

Flowchart of if-else statement in C



Let's see the simple example of even and odd number using if-else statement in C language.

```
#include<stdio.h>
void main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
else{
printf("%d is odd number",number);
}
getch();;
}
```

Output

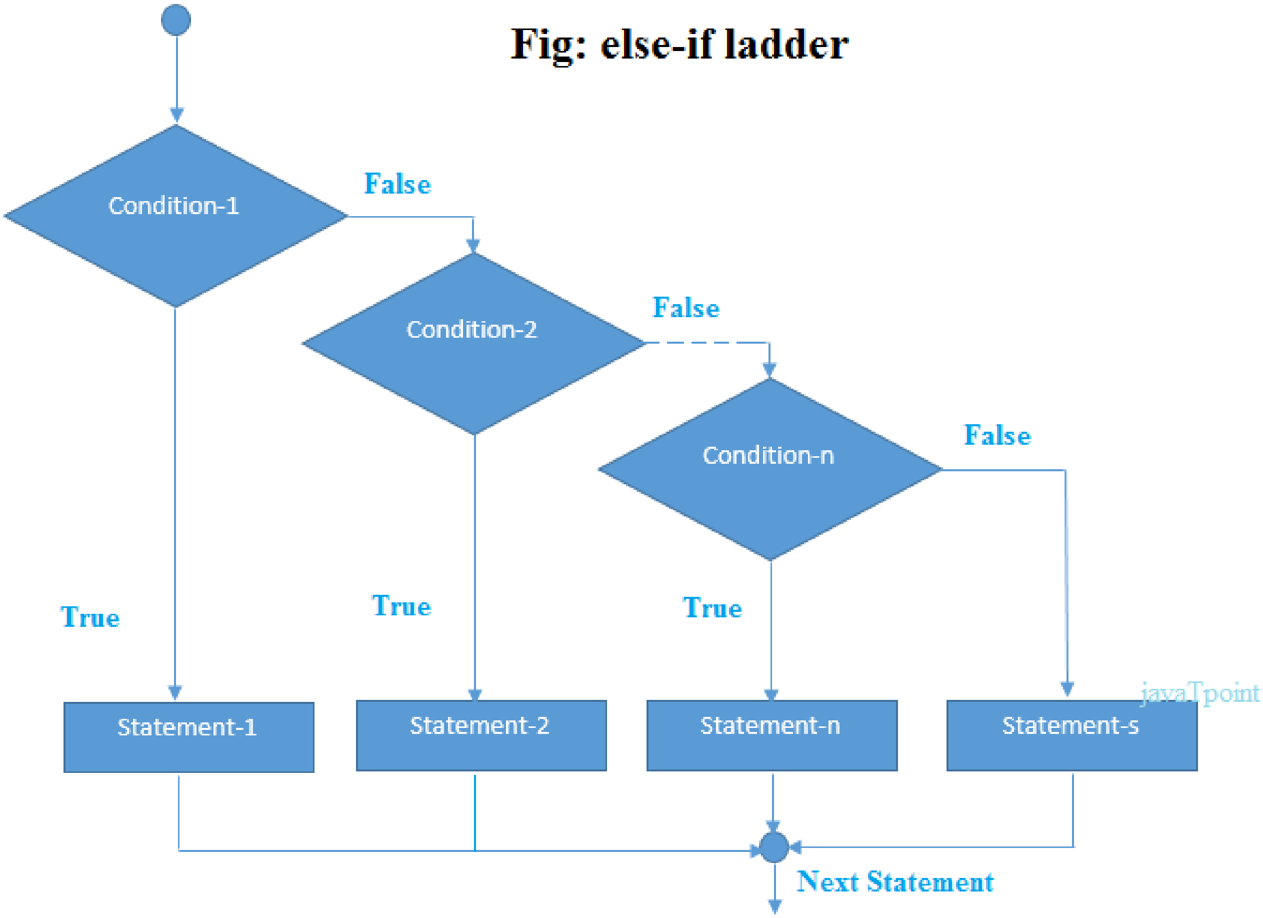
```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

If else-if ladder Statement

The if else-if statement is used to execute one code from multiple conditions. The syntax of if else-if statement is given below:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Flowchart of else-if ladder statement in C



The example of if-else-if statement in C language is given below.

```
#include<stdio.h>
void main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10){
printf("number is equals to 10");
}
else if(number==50){
printf("number is equal to 50");
}
else if(number==100){
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");
}
getch();;
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Switch Statement

The switch statement in C language is used *to execute the code from multiple conditions*. It is like if else-if ladder statement.

The syntax of switch statement in c language is given below:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......
 default:
 code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

1) The *switch expression* must be of integer or character type.

2) The *case value* must be integer or character constant.

3) The *case value* can be used only inside the switch statement.

4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in switch case, all the cases will be executed after matching the case value. It is known as *fall through* state of C switch statement.

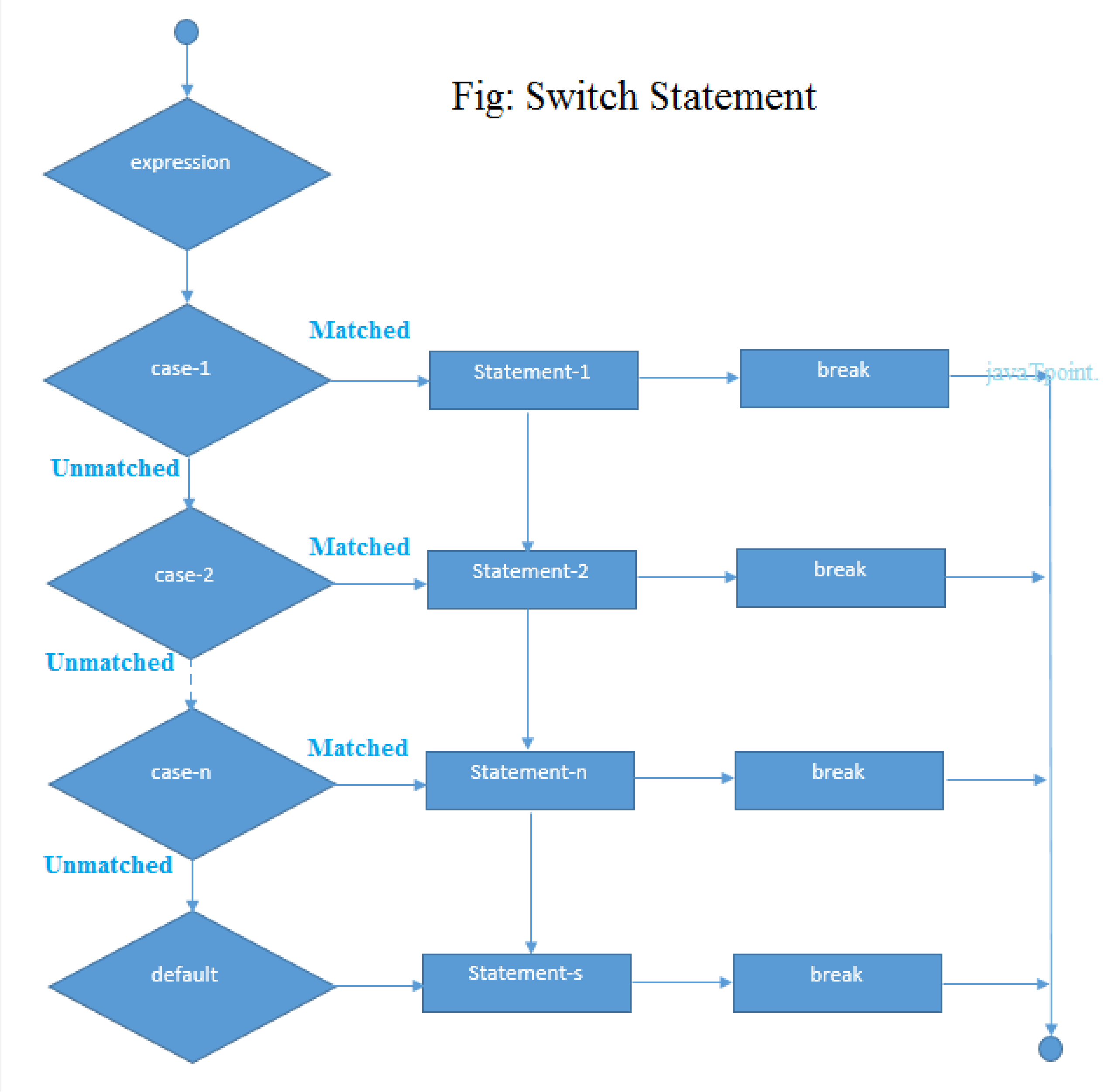
Let's try to understand it by the examples. We are assuming there are following variables.

```
int x,y,z;
char a,b;
float f;
```

Valid Switch	Invalid Switch	Valid Case	Invalid Case

switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Flowchart of switch statement in C



Let's see a simple example of c language switch statement.

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

```
void main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
getch();
}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Loops

The *loops in C language* are used to execute a block of code or a part of the program several times.

In other words, it iterates a code or group of code many times.

Why use loops in C language?

Suppose that you have to print table of 2, then you need to write 10 lines of code.

By using the loop statement, you can do it by 2 or 3 lines of code only.

Advantage of loops in C

1) It **saves code**.

2) It helps to traverse the elements of array (which is covered in next pages).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while

2. while
3. for

do while loop in C

To execute a part of program or code several times, we can use do-while loop of C language. The code given between the do and while block will be executed until condition is true.

In do while loop, statement is given before the condition, so *statement or code will be executed at lease one time*. In other words, we can say it is executed 1 or more times.

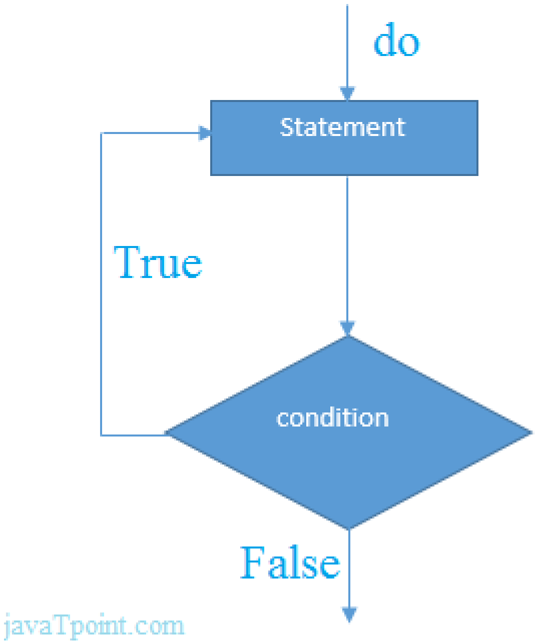
It is better if you have to execute the code at least once.

do while loop syntax

The syntax of C language do-while loop is given below:

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

Flowchart of do while loop



do while example

There is given the simple program of c language do while loop where we are printing the table of 1.

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

```
int i=1;
do{
printf("%d \n",i);
i++;
}while(i<=10);
getch();
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

while loop in C

The **while loop** in C language is *used to iterate the part of program or statements many times*.

In while loop, condition is given before the statement. So it is different from the do while loop. It can execute the statements 0 or more times.

When use while loop in C

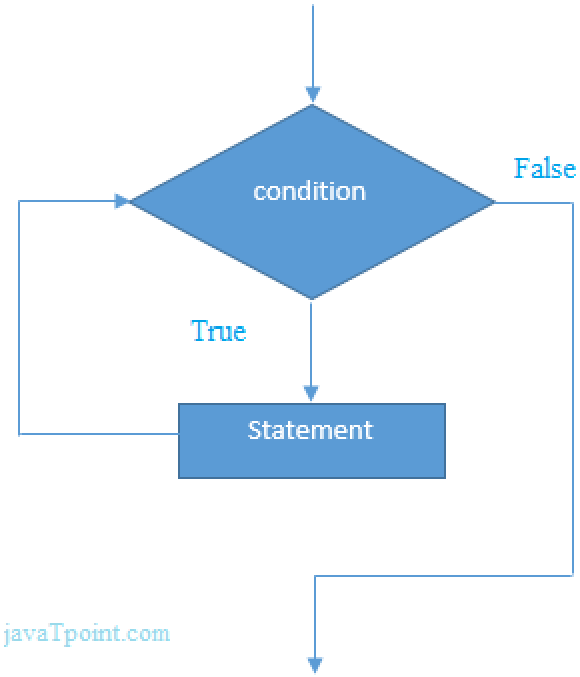
The C language while loop *should be used if number of iteration is uncertain or unknown*.

Syntax of while loop in C language

The syntax of while loop in c language is given below:

```
while(condition){
//code to be executed
}
```

Flowchart of while loop in C



Example of while loop in C language

Let's see the simple program of while loop that prints table of 1.

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

Output

```
1
2
3
4
5
6
7
8
9
10
```

for loop in C

The **for loop in C language** is also *used to iterate the statement or a part of the program several times*, like while and do-while loop.

But, we can initialize and increment or decrement the variable also at the time of checking the condition in for loop.

Unlike do while loop, the condition or expression in for loop is given before the statement, so it may execute the statement 0 or more times.

When use for loop in C

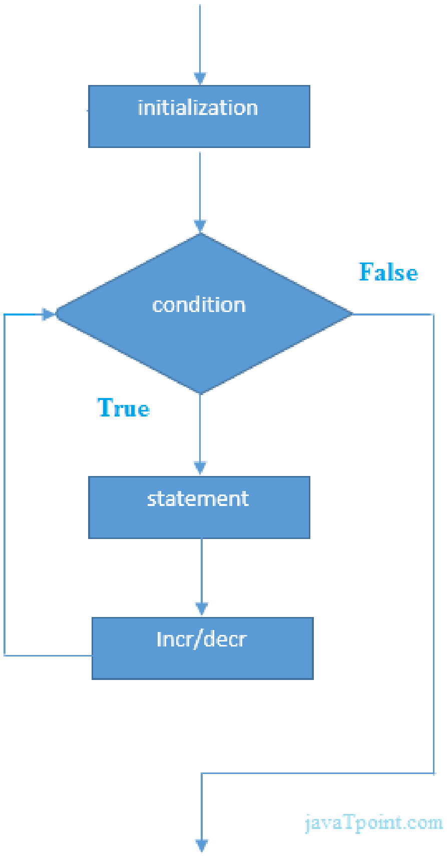
For loop is better if number of iteration is known by the programmer.

Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

Flowchart of for loop in C



Example of for loop in C language

Let's see the simple program of for loop that prints table of 1.

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

Output

```
1
2
3
4
5
6
7
8
9
10
```

C break statement

The **break statement** in C language is used to break the execution of loop (while, do while and for) and switch case.

In case of *inner loops*, it terminates the control of inner loop only.

There can be two usage of C break keyword:

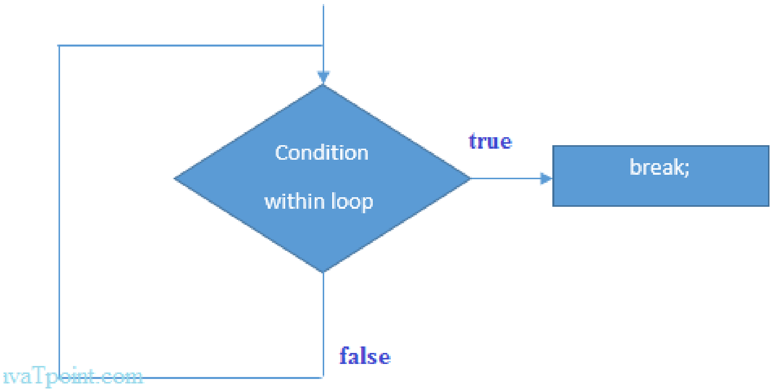
1. With switch case
2. With loop

Syntax:

```
jump-statement;
break;
```

The jump statement in c break syntax can be while loop, do while loop, for loop or switch case.

Flowchart of break in c



Example of C break statement with switch case

Click here to see the example of C break with switch statement.

Example of C break statement with loop

```
#include<stdio.h>
void main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
printf("%d \n",i);
if(i==5){//if value of i is equal to 5, it will break the loop
break;
}
}
getch();
}
```

Output

```
1
2
3
4
5
```

C continue statement

The **continue statement** in C language is used to continue the execution of loop (while, do while and for). It is used with *if condition* within the loop.

In case of *inner loops*, it continues the control of inner loop only.

Syntax:

```
jump-statement;
```

```
continue;
```

The *jump statement* can be while, do while and for loop.

Example of continue statement in c

```
#include<stdio.h>
void main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5){//if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d \n",i);
}//end of for loop
getch();
}
```

Output

```
1
2
3
4
6
7
8
9
10
```

C goto statement

The goto statement is known as jump statement in C language. It is used to unconditionally jump to other label. It transfers control to other parts of the program.

It is rarely used today because it makes program less readable and complex.

Syntax:

```
1. goto label;
```

goto example

Let's see a simple example to use goto statement in C language.

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

```
    ineligible:
    printf("You are not eligible to vote!\n");

    printf("Enter you age:\n");
    scanf("%d", &age);
    if(age<18)
        goto ineligible;
    else
        printf("You are eligible to vote!\n");

  getch();
  }
```

Output:

```
You are not eligible to vote!
Enter you age:
11
You are not eligible to vote!
Enter you age:
44
You are eligible to vote!
```

Storage Classes in C

Storage classes are used to define scope and life time of a variable. There are four storage classes in C programming.

- auto
- extern
- static
- register

Storage Classes	Storage Place	Default Value	Scope	Life-time
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of main program, May be declared anywhere in the program
static	RAM	Zero	Local	Till the end of main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within function

1) auto

The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable.

```
#include<stdio.h>
void  main(){
int a=10;
auto int b=10;//same like above
printf("%d %d",a,b);
getch();
}
```

Output:

```
10 10
```

2) register

The register variable allocates memory in register than RAM. Its size is same of register size. It has a faster access than other variables.

It is recommended to use register variable only for quick access such as in counter.

Note: We can't get the address of register variable.

```
register int counter=0;
```

3) static

The **static** variable is initialized only once and exists till the end of the program. It retains its value between multiple functions call.

The static variable has the default value 0 which is provided by compiler.

```
#include<stdio.h>
int  func(){

   static int i=0;//static variable
   int j=0;//local variable
   i++;
   j++;
   printf("i= %d and j= %d\n", i, j);
}
void main() {
  func();
  func();
  func();
getch();
}
```

Output:

```
i= 1 and j= 1
i= 2 and j= 1
i= 3 and j= 1
```

4) extern

The extern variable is visible to all the programs. It is used if two or more files are sharing same variable or function.

1. `extern int counter=0;`

Type Casting in C

Type casting allows us to convert one data type into other. In C language, we use cast operator for type casting which is denoted by (type).

Syntax:

1. `(type)value;`

> ***Note: It is always recommended to convert lower value to higher for avoiding data loss.***

Without Type Casting:

```
int f= 9/4;
printf("f : %d\n", f );//Output: 2
```

With Type Casting:

```
float f=(float) 9/4;
printf("f : %f\n", f );//Output: 2.250000
```

Type Casting example

Let's see a simple example to cast int value into float.

```
#include<stdio.h>
void main(){
float f= (float)9/4;
printf("f : %f\n", f );
getch();
}
```

Output:

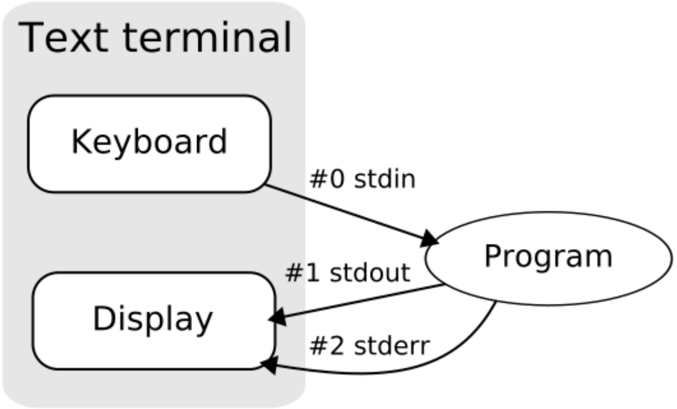
```
f : 2.250000
```

File I/O Streams in C Programming Language :

1. In C all **input and output** is done with streams
2. Stream is nothing but the **sequence of bytes of data**
3. A sequence of bytes flowing into program is called **input stream**
4. A sequence of bytes flowing out of the program is called **output stream**
5. Use of Stream make I/O machine independent.

Predefined Streams :

stdin	**Standard Input**
stdout	Standard Output
stderr	Standard Error



Standard Input Stream Device :

1. **stdin** stands for (**St**an**d**ard **In**put)
2. Keyboard is **standard input device** .
3. Standard input is data **(Often Text) going into a program**.
4. The program requests data transfers by use of the read operation.
5. Not all programs require input.

Standard Output Stream Device :

1. **stdout** stands for (**St**an**d**ard **Out**put)
2. Screen(Monitor) is **standard output device** .
3. Standard output is data **(Often Text) going out from a program**.

4. The program sends data to output device by using write operation.

Difference Between Std. Input and Output Stream Devices :

Point	**Std i/p Stream Device**	**Standard o/p Stream Device**
Stands For	Standard Input	Standard Output
Example	Keyboard	Screen/Monitor
Data Flow	Data (Often Text) going into a program	data (Often Text) going out from a program
Operation	Read Operation	Write Operation

Some Important Summary :

Point	**Input Stream**	**Output Stream**
Standard Device 1	Keyboard	Screen
Standard Device 2	Scanner	Printer
IO Function	scanf and gets	printf and puts
IO Operation	Read	Write
Data	Data goes from stream	data comes into stream

Command Line Arguments in C

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program.

To support command line argument, you need to change the structure of main() function as given below.

```
int main(int argc, char *argv[] )
```

Here, **argc** counts the number of arguments. It counts the file name as the first argument.

The **argv[]** contains the total number of arguments. The first argument is the file name always.

Example

Let's see the example of command line arguments where we are passing one argument with file name.

```
#include <stdio.h>
void main(int argc, char *argv[] )  {

   printf("Program name is: %s\n", argv[0]);

   if(argc < 2){
      printf("No argument passed through command line.\n");
   }
   else{
      printf("First argument is: %s\n", argv[1]);
   }
}
```

Run this program as follows in Linux:

```
./program hello
```

Run this program as follows in Windows from command line:

```
program.exe hello
```

Output:

```
Program name is: program
First argument is: hello
```

If you pass many arguments, it will print only one.

```
./program hello c how r u
```

Output:

```
Program name is: program
First argument is: hello
```

But if you pass many arguments within double quote, all arguments will be treated as a single argument only.

```
./program "hello c how r u"
```

Output:

```
Program name is: program
First argument is: hello c how r u
```

You can write your program to print all the arguments. In this program, we are printing only argv[1], that is why it is printing only one argument.

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

int main(int argc, char *argv[])   //  command line arguments
{
if(argc!=5)
{
   printf("Arguments passed through command line " \
          "not equal to 5");
   return 1;
}

   printf("\n Program name  : %s \n", argv[0]);
   printf("1st arg  : %s \n", argv[1]);
   printf("2nd arg  : %s \n", argv[2]);
   printf("3rd arg  : %s \n", argv[3]);
   printf("4th arg  : %s \n", argv[4]);
   printf("5th arg  : %s \n", argv[5]);

return 0;
}
```

```
OUTPUT:Program name : test
1st arg : this
2nd arg : is
3rd arg : a
4th arg : program
5th arg : (null)
```