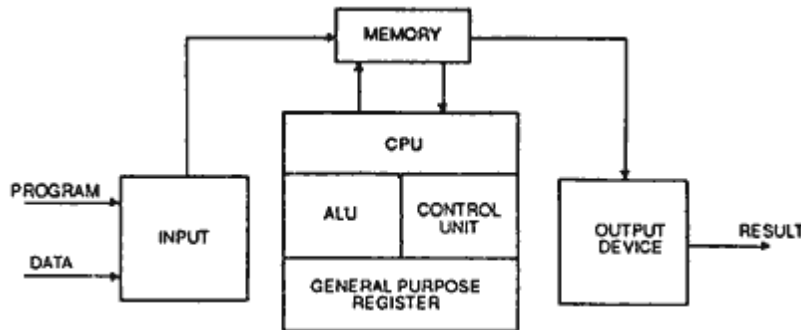


This document is created for beginners for basic reference who wanted to learn from scratch, everyone wants to learn X programming language by this semester/year there is no way you can sit with a book and learn a language. You need to do some real work with it, develop some real software and not just do those exercises in the book (that is necessary of course but not sufficient). Most of the languages I have learnt are because I was forced to do so as part of some project. Just pick up the basics in a day or two and then apply it to a real life project. Need ideas? Come to us.

INTRODUCTION

A Computer
electronic
instructions
input
data
instructions
results to



is programmable
machine that accepts
and data through
devices, manipulates
according to
and finally provides
output devices.

1. First Generation Computers

- ✓ Vacuum tubes were used which produce more heat
- ✓ Speed of computing was measured in milliseconds
- ✓ Limited storage capacity

2. Second – Generation Computers

- ✓ Transistors and diodes were used.
- ✓ Speed of computing was measured in microseconds
- ✓ Storage capacity was increased
- ✓ Magnetic tapes were used instead of punching cards.

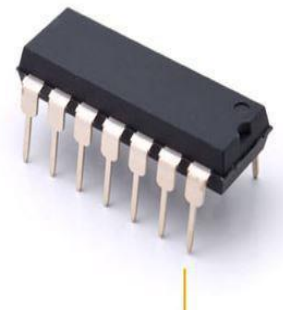
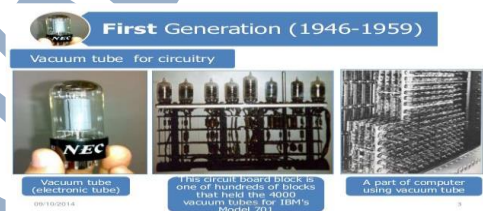
3. Third Generation Computers

- ✓ Integrated Circuits were used.
- ✓ Speed is measured in nanoseconds
- ✓ Occupied less space.
- ✓ Devices like visual display unit for I/O devices.

4. Fourth – Generation Computers

- ✓ Use of micro processor chip
- ✓ Speed was measured in nano and picoseconds
- ✓ Occupied less space
- ✓ Commonly available as personal computers
- ✓ Mini & micro Computers are developed from micro-processor

5. Fifth – Generation Computers:



- ✓ Use of super large-scale integration (SLSI) chip in computer (super computers)
- ✓ Capable of performing millions of instructions per seconds (MIPS)

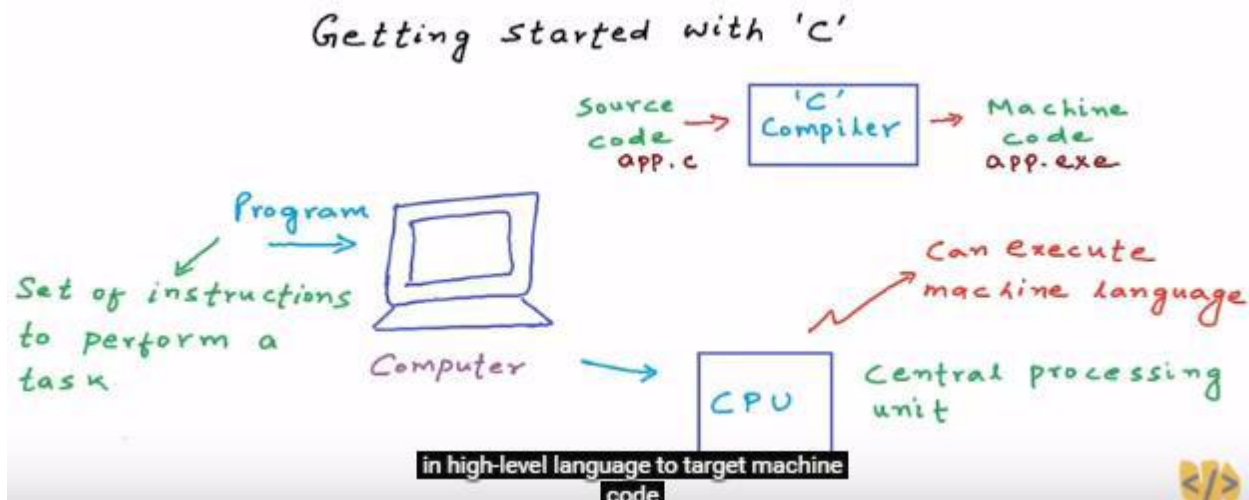
Types of Computers:

- ✓ Super computers:
These are specialized and task specific computers used by large organizations. These computers are used for research and exploration purposes, like NASA uses supercomputers for launching space shuttles, controlling them and for space exploration purpose.
- ✓ Mainframe Computers:
Mainframes can also process & store large amount of data. Banks educational institutions & insurance companies use mainframe computers to store data about their customers, students & insurance policy holders.
- ✓ Mini Computers:
Minicomputers are used by small businesses & firms. Minicomputers are also called as “Midrange Computers”. These computers are not designed for a single user. Individual departments of a large company or organizations use Mini-computers for specific purposes.
- ✓ Micro Computers:
Desktop computers, laptops, personal digital assistant (PDA), tablets & smart phones are all types of microcomputers. The micro-computers are widely used & the fastest growing computers.

Data Storage in a Computer

- ✓ 4bits = 1 Nibble
- ✓ 8bits = 1 byte
- ✓ 1024 bytes = 1k or 1kb (kilobyte)
- ✓ 1024KB = 1MB (mega byte)
- ✓ 1024MB = 1GB (Giga byte)
- ✓ 1024GB = 1TBC Terabytes

A computer is general purpose machine that can perform any computational task for us. To perform a task you need to give it a program which is nothing but set of instructions to perform the task, the core part of the computer that executes these instructions is called central processing unit or CPU. The CPU can execute the instructions only in what we call machine language which is binary encoded as per some rules machine language is called low-level programming language It's too cryptic to write and it varies from one architecture to another but unfortunately we can write a program in what we call high level language like C.



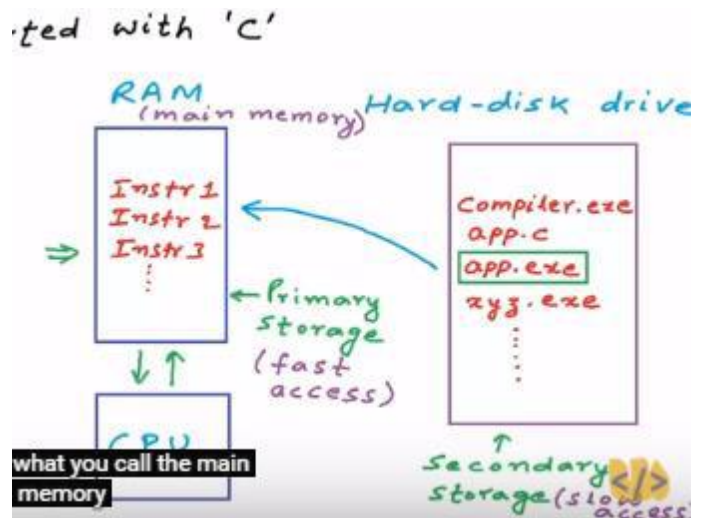
Even shortcut on desktops is executable file. All files in computers are typically stored in hard disk drive. Compiler itself is a program so it will have an executable then you can have all kinds of files. If a program is there on your hard disk it does not mean that it will be always running to run the program you need to double click on executable file or nice little icon that will be shortcut to the executable file.

Now program has to run. Now the instructions in the program have to get executed what happens is that when you run the program. Let's say you ran app.exe the program or the set of instructions is transferred to what we call random access memory (RAM) is volatile memory everything is wiped off is cleared as power off's. But anything in hard disk remains constant.

Ram is primary storage & hard disk is secondary storage. When instructions are loaded in ram, they are fetched sequentially by CPU. CPU is guy who executes the instructions. CPU can read from the RAM and write back to RAM. CPU has less memory very less its small chip. But good thing CPU can execute million instructions per second.

But u might get a doubt? Why we need RAM? This is because access to RAM is faster than hard disk performance and time is good.

When program finishes executing finishes program gets cleared. A number of programs all resources are shared by RAM n CPU are managed by Operating system.



Software:

Software is a collection of programs. A program consists of set of instructions which is designed to perform a particular task.

N no of programs together combined to forms a software tool or component.

Software is divided into two categories:

- ✓ Application software

- ✓ System software

System software:

The software designed for general purpose & doesn't contain any limitations.

Ex: OS is an interface between hardware component and software.

Application software:

Software which is designed for specific task.

Ex: Oracle, Ms-Office, Tally

Oracle is application software that is used to maintain data in tabular format.

Ms Office is application software that is used to maintain data in document format.

Tally is software that is used to maintain account related information.

C programming is a popular programming language used for creating system and application software.

Programming language:

It is a special kind of instructions which is used to communicate with computers.

Among computer generations we have 3 kinds of languages.

1. Machine Language
2. Assembly Level language
3. High Level Language

High-Level Language	Assembly Language	Machine Language
a + b	ld [%fp-20], %o0 ld [%fp-24], %o1 add %o0, %o1, %o0	... 1101 0000 0000 0111 1011 1111 1110 1000 1101 0010 0000 0111 1011 1111 1110 1000 1001 0000 0000 0000 ...

1. Machine Level Language:

In this language all the instructions are written in 0's and 1's only.

- ✓ It is not easy to learn and write the programs.
- ✓ Not understood by any other users.
- ✓ Performed by only experts.

So to make the programming easy new language is invented.

2. Assembly Level language:

In this Language some kind of English words with machine code is used. The English words are referred as "MNEMONICS Code", and they are like ADD, SUB, MUL, DIV, LOAD, SAVE etc.....

But system can understand only the machine language.

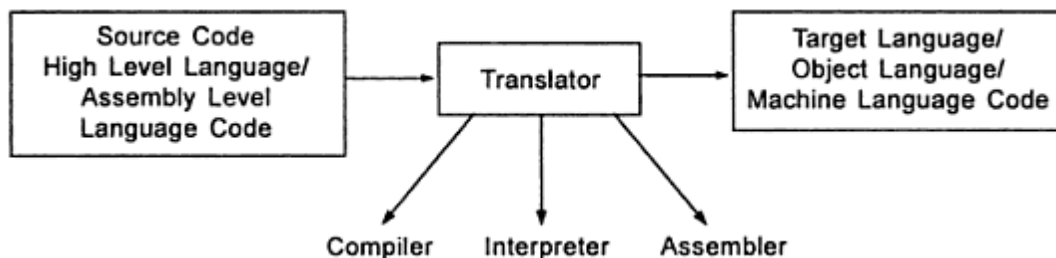
To convert the "MNEMONICS" into machine language a converter is required. Here the converter is "Assembler". It converts the assembly language into Machine level language

The program written in high level language is source code and compiled program is object code. Compilers are used to convert the high level languages into machine code and machine language into high level language.

It converts the entire high level code into machine code at a time and if the code contains any syntax errors then it reports at the end of the compilation.

Interpreter:

This is also used to convert the high level program into machine code. But it checks for errors, statement by statement and converts into machine code if the statement is correct. If an error is occurred, then the program conversion stops at the erroneous statement. It is slow process when comparing with compiler.



History of C

Any programming language is an interface to Computer. Three important aspects of a language are the way it stores data, how it accomplishes input and output operations it uses to transform and combine data.

From 1960 a hoarde of computer languages had come into existence, almost each language has its own specific purpose. For example COBOL - Commercial applications, FORTRAN was used for engineering & scientific applications. So people thought instead of learning so many languages. Why not use one language which can perform all possible applications. Then committee came out with algol60 but not was not popular then CPL developed it was hard to implement & learn. Later BCPL & B came.

In 1967's there was a programming language called B.C.P.L (Basic combined programming language) invented by "Martin Richards". In 1970's a person named as 'Ken Thompson' made some changes in B.C.P.L and created a new language called as 'B-Language' by taking first letter from B.C.P.L. Later on a person 'Dennis Ritchie' re-modified the 'B- Language' and created a new language. He named this language as 'C- language' by taking second letter from 'B.C.P.L.

C Programming is an ANSI/ISO standard and powerful programming language for developing real time applications. C programming language was invented by Dennis Ritchie at the Bell Laboratories in 1972. After that by using 'C-Language' he re-designed the 93% UNIX operating system. From than 'C' is branded as OPERATING SYSTEM programming language. C is most widely used programming language even today. Many C compilers, Microsoft adopted C developed visual c++. All other programming languages were derived directly or indirectly from C programming concepts. Standardization process was done by 1990 in ANSI/ISO committee or ANSI/ISO C.

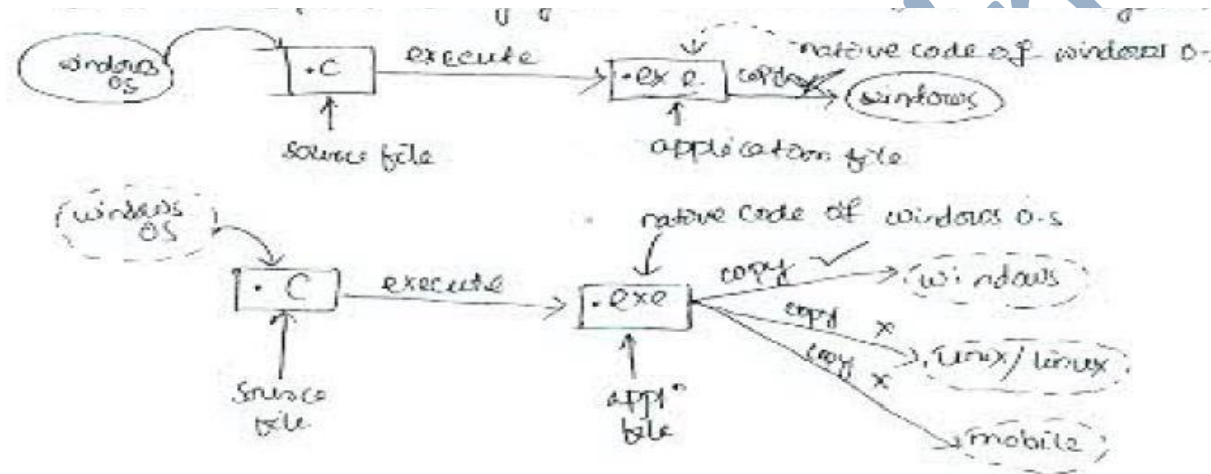
In order to run a C program, you need a compiler. Compiler change source code (code written by programmer) to object code (code that computer understands) and creates executable file. There are many free and professional compilers available. For the sake of this course, GNU GCC compiler is used.

A compiler turns the program that you write into an **executable** that your computer can actually understand and run. If you're on Linux, you can use gcc, and if you're on Mac OS X, you can use Code.

As per below observation when we are copying .exe file to any other system which contains windows OS then it works properly, because native code of the application. Same .exe file when we are copying to any other operating system like Unix/Linux then it doesn't work because native code is different.

C programming is platform dependent and machine independent programming language C doesn't depends on hardware components of the system.

C is also called as mid level language because it supports high-level instructions with the combination of low level programming also.



Applications of C:

System Software designing i.e. OS and compilers

Application software designing i.e. database and excel sheets

Graphic related application i.e. PC games and mobile games.

A C program can vary from 3 lines to millions of lines and it should be written into one or more text files with extension ".c";

Algorithm

An Algorithm is called pseudocode (language independent code)

Algorithm is step by step instruction of performing any task or solves any problem or

It is series of steps in logical sequence to solve a problem.

Each step is called Instruction.

Algorithm must satisfy five properties:

- ✓ Input: zero or more inputs
- ✓ Output: one or more outputs
- ✓ Definiteness: Algorithm should be clear and unambiguous.
- ✓ Finiteness: the algorithm should terminate after finite no of steps.
- ✓ Effectiveness: All operations performed exactly in fixed duration of time.

There are several different algorithms for sorting 'n' no's in such situation we should pick a best or most efficient algorithm.

Problem 1: Find the area of a Circle of radius r.

Input to the algorithm: Radius r of the Circle.

Expected output: Area of the Circle

Algorithm:

- Step1: Read\input the Radius r of the Circle
- Step2: Area $\pi * r * r$ // calculation of area
- Step3: Print Area

Problem2: Write an algorithm to read two numbers and find their sum.

Inputs to the algorithm:

First num1. Second num2.

Expected output:

Sum of the two numbers.

Algorithm:

- Step1: Start
- Step2: Read\input the first num1.
- Step3: Read\input the second num2.
- Step4: Sum $\text{num1} + \text{num2}$ // calculation of sum
- Step5: Print Sum
- Step6: End

Question:

What is the significance of an algorithm to C programming?

Before a program can be written, an algorithm has to be created first. An algorithm provides a step by step procedure on how a solution can be derived. It also acts as a blueprint on how a program will start and end, including what process and computations are involved.



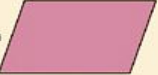

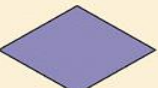
Flow Chart:

It is pictorial representation of a sequence of instructions using predefined symbols.

A flowchart uses different symbols or shapes or boxes symbols to denote different types of instructions these symbols are connected by solid lines with arrow marks to indicate the operation flow.

A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- E.g. control flow from one action to the next

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g., PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).

Example: Algorithm is represented in form of a flow chart & that flowchart is then represented in form of some programming language to prepare computer program.

Program:

It is set of instructions written in any programming language to solve a problem using computer.

C program structure:

Documentation Section

Link Section

Definition Section

Global declaration Section

main() { //function

Declaration part

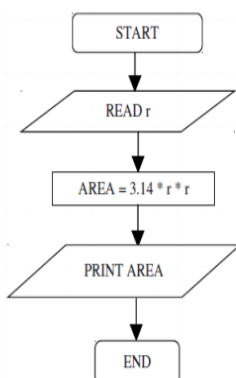
Executable part

}

Sub program section

- ✓ Documentation section: Consists of comment lines giving the name of program. Details of program. There are single line comments,
- ✓ Link section: provides instructions to the compiler to link function from the system library (ex:stdio.h or conio.h)

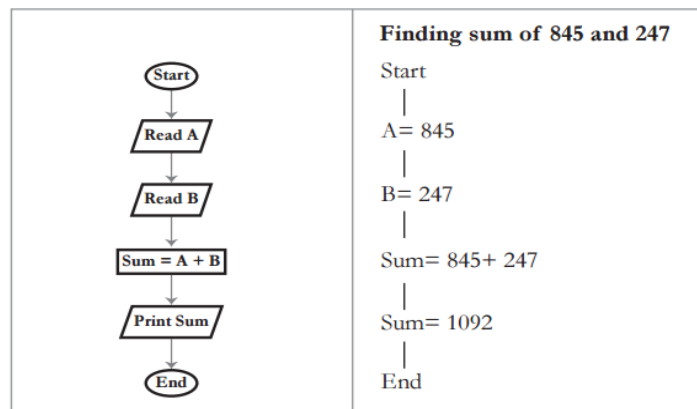
Find the area of a circle of radius r.



✓ Definition section: defines all symbolic constants

✓ Global declaration section: variables that are used in more than one function are

Flowchart - How to find sum of two numbers



called global variables are declared in global declaration section.

Every C program must have one main() function contains two parts declaration part and executable part. All declaration and executable parts ends with semicolon (;).

- ✓ Declaration part declares all the variables used in executable part. These two parts must appear in between {and} closing braces.
- ✓ Execution starts at main {and ends with} closing brace is logical end part of the program.
- ✓ Sub program section: all user defined functions that are called from main function. Except main function section, all may be absent.
- ✓ #include<stdio.h> this is a linking section here we are telling to compiler what header files should link our program without linking of header files corresponding predefined functions won't work. In this program predefined functions are printf & scanf (these are existing in header file or library)

#include is preprocessor directive.

The preprocessor is program that processes the source program before it is passed on compiler. The preprocessor directives are generally initialized at the beginning of a program it begins with #

Ex: #define PI 3.14

main() - is a special function starting point for program execution & only exactly one main function should be there in program. If we are trying to use more than one compiler doesn't know which is beginning of the program.

() -- indicates no parameters or arguments.

{ -- beginning of func main

}-- indicates end of function

All statements should be inside function body.

Remember C language is case sensitive.

Each instruction in a C program is written as a separate statement. Therefore a complete C program would comprise of a series of statements. In C everything is written using lowercase letter.

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.

Even you can place comment after statement. For example:

```
printf("lkgtopg.in");//printing information
```

Multi 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
*/
```

Before we study the basic building blocks of the C programming language, let us look at a bare minimum C program structure

```
#include <stdio.h> // header file
int main() {
/* my first program in C */
printf("Hello, from lkgtopg.in\n");
return 0;
}
```

Let us take a look at the various parts of the above program –

- ✓ The first line of the program `#include <stdio.h>` is a preprocessor command, which tells a C compiler to include `stdio.h` file includes standard input output header file(`stdio.h`) from the C library before compiling a C program.
- ✓ The next line `int main()` is the main function where the program execution begins.
- ✓ { This indicates the beginning of the main function.
- ✓ The next line `/*...*/` will be ignored by the compiler whatever is given inside the command `“/*...*/”` in any C program, won't be considered for compilation and execution.
- ✓ The next line `printf(...)` is another function available in C which causes the message " Hello, from lkgtopg.in" to be displayed on the screen. This is used to display something on console/screen.
- ✓ The next line **return 0;** This command terminates C program (main function) and returns 0..
} This indicates the end of the main function.

Question:

What is the difference between `<stdio.h>` and `"stdio.h"`?

For `#include "filename"` the preprocessor searches in the same directory as the file containing the directive. This method is normally used to include programmer-defined header files.

For `#include <filename>` the preprocessor searches in an implementation dependent manner, normally in search directories pre-designated by the compiler/IDE. This method is normally used to include standard library header files i.e. **searches in standard C library locations.**

C – Data Types

C data types are defined as the data storage format that a variable can store a data to perform a specific operation.

Data types are used to define a variable before to use in a program.

Size of variable, constant and array are determined by data types

There are four data types in C language. They are,

S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	enum
3	Derived data type	pointer, array, structure, union
4	Void data type	void

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

int: As the name suggests, an int variable is used to store an integer.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

Declaration of Variable: It tells the compiler what the variable name is used, what type of data is held by the variable.

A variable is an identifier (or) user defined word. Its value is changed during program execution. A variable is the name of the memory location at where we are going to store data. Different types of variables require different amounts of memory.

Syn: datatype v1,v2,...vn;

Eg : int a, b;

float sum;

double ratio;

Initializing variables:

When variables are declared, values can be assigned to them in two ways.

// declaration and definition of variable abc

char abc = 'a';

// multiple declarations and definitions

int _a, _b, c,d;

Within a type declaration: the value is assigned at the declaration time

Example: int a=10;

float b=3.14;

char c='a';

Using assignment statement: the value is assigned after declarations are made

Example: int a;

float b;

char c;

a=10;

b=3.14;

c='a';

Program to declare a variable and initialize it

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

```
main() {
```

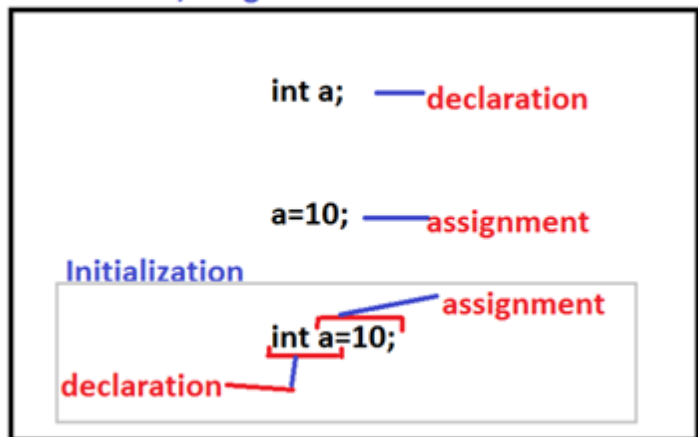
```
int a=0; // a is declared here
```

```
a is initialized with 0
```

```
printf("%d",a);
```

```
}
```

Declaration, Assignment & Initialization



Common Mistake: (Missing Semicolon)

Every statement must end with a semicolon. A missing semicolon may cause considerable confusion to the compiler and result in misleading error messages.

Consider the following statements.

```
a = d*e
```

```
b = j/k;
```

The compiler will treat the second line as a part of the first one and treat b as a variable name.

Test your skill:

```
int var;
```

```
int main() {
```

```
var = 10;
```

```
return 0;
```

```
}
```

Explanation: This program is compiled successfully. Here var is defined (and declared implicitly) globally.

```
extern int var;
```

```
int main(void) {
```

```
return 0;
```

```
}
```

Explanation: This program is compiled successfully. Here var is declared only. Notice var is never used so no problems.

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

```
int var = 20;
```

```
int main(){
```

```
int var = var;
```

```
printf("%d ", var);
```

```
return 0;
```

```
}
```

(A) Garbage Value

(B) 20

(C) Compiler Error

Answer: (A)

Explanation: First var is declared, then value is assigned to it. As soon as var is declared as a local variable, it hides the global variable var.

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

```
int main() {
```

```
int i;
```

```
i = 1, 2, 3;
```

```
printf("%d", i);
```

```
return 0;
```

```
}
```

(A) 1

(B) 3

(C) Garbage value

(D) Compile time error

Answer: (A)

Explanation: Comma acts as an operator. The assignment operator has higher precedence than comma operator. So, the expression is considered as (i = 1), 2, 3 and 1 gets assigned to variable i.

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

```
int main(){
```

```
int i = 1, 2, 3;
```

```
printf("%d", i);
```

```
return 0;
```

```
}
```

(A) 1

(B) 3

(C) Garbage value

(D) Compile time error

Answer: (D)

Explanation: Comma acts as a separator here. The compiler creates an integer variable and initializes it with 1. The compiler fails to create integer variable 2 because 2 is not a valid identifier.

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

```
int main(){
```

```
int i = (1, 2, 3);
```

```
printf("%d", i);
```

```
return 0;
```

```
}
```

(A) 1

(B) 3

(C) Garbage value

(D) Compile time error

Answer: (B)

Explanation: The bracket operator has higher precedence than assignment operator. The expression within bracket operator is evaluated from left to right but it is always the result of the last expression which gets assigned.

```
int main()
```

```
{
```

```
int x, y = 5, z = 5;
```

```
x = y == z;
```

```
printf("%d", x);
```

```
getchar();
```

```
return 0;
```

```
}
```

(A) 0

(B) 1

(C) 5

(D) Compiler Error

Answer: (B)

Explanation: The crux of the question lies in the statement `x = y == z`. The operator `==` is executed before `=` because precedence of comparison operators (`<=`, `>=` and `==`) is higher than assignment operator `=`.

The result of a comparison operator is either 0 or 1 based on the comparison result. Since y is equal to z, value of the expression `y == z` becomes 1 and the value is assigned to x via the assignment operator.

Output:

```
int main(){
int goto=20;
printf("%d",goto);
return 0;
}
```

Explanation: Compilation Error as goto is a keyword. Invalid variable name. goto is keyword in c. variable name cannot be any keyword of c language.

Output:

```
int main(){
long int 1a=5l;
printf("%ld",1a);
return 0;
}
```

Explanation: Invalid variable name. Variable name must start from either alphabet or underscore. but not using number.

Output:

```
#include<stdio.h>
int main(){
int max-val=100;
int min-val=10;
printf("%d",max-val);
printf("%d",min-val);
return 0;
}
```

Explanation:

We cannot use special character – in the variable name.

Output:

```
int main(){
int abcdefghijklmnopqrstuvwxyz123456789=10;
int abcdefghijklmnopqrstuvwxyz123456=40;
printf("%d",abcdefghijklmnopqrstuvwxyz123456);
return 0;
}
```

Explanation: Only first 32 characters are significant in variable name. So compiler will show error: Multiple declaration of identifier abcdefghijklmnopqrstuvwxyz123456.

```
int main()
{
int x = 032;
printf("%d", x);
return 0;
}
```

}

Explanation: When a constant value starts with 0, it is considered as octal number. Therefore the value of x is $3*8 + 2 = 26$

Common Mistake: Undeclared Variables

Every variable must be declared for its type, before it is used.

Output:

What will be output when you will execute following c code?

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

```
int main(){
```

```
printf("%d",sizeof(10.5));
```

```
printf("%d",sizeof(60000));
```

```
printf("%d",sizeof('O')); return 0;
```

```
}
```

Explanation: 8 4 2 (Depends on compiler)

By default data type of numeric constants is:

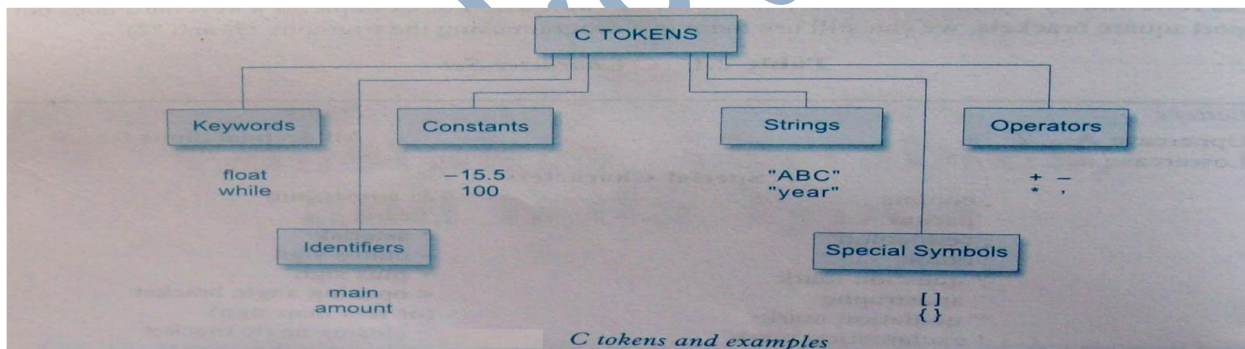
10.5 : double

60000: long int

'O': char

Algebraic Expression	C Expression
$a \times b - c \times d$	$a * b - c * d$
$(m + n) (a + b)$	$(m + n) * (a + b)$
$3x^2 + 2x + 5$	$3 * x * x + 2 * x + 5$
$\frac{a + b + c}{d + e}$	$(a + b + c) / (d + e)$
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)} \right]$	$2 * b * y / (d + 1) - x / 3 * (z + y)$

C Tokens



In C program, the smallest individual units are known as C tokens

C has six types of tokens

Every C program is written using these tokens and the syntax of the language

Delimiters

These are the special symbols and they have a special task to use in your programs.

#(hash): preprocessor directive:

At the time of compilation the code for the predefined functions are verified and transformed into machine language. This is a preprocess and the statements that invokes this kind of process are treated as preprocessor directives.

In 'C' we have 3 preprocess directives

#include => for header file specification

#define => for macro specification

#if => for condition testing

;(semicolon):

=====

this used to end the statement in the programming

:(colon)

=====

to terminate the labels.

,(comma):

=====

commas are used to separate variables.

[(square brackets):

=====

square brackets are used at array declaration.

< >(angle brackets):

=====

angle brackets are used at the header file specification.

)(parenthesis):

=====

parenthesis is used at the function declaration and for the expression indication.

{ }(curly braces):

=====

This indicates the block declaration.

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

```
int main(){
```

```
char a = '\012';
```

```
printf("%d", a);
```

```
return 0;
```

```
}
```

The value '\012' means the character with value 12 in octal, which is decimal 10

What is the use of a semicolon (;) at the end of every program statement?

It has to do with the parsing process and compilation of the code. A semicolon acts as a delimiter i.e. ends of instruction, so that the compiler knows where each statement ends, and can proceed to divide the statement into smaller elements for syntax checking.

Operators

An operator is a symbol that tells the computer to perform certain mathematical or logical operations. Operators are used in programs to manipulate data and variables. In the below example “+” is called as “operator”.

An expression is a sequence of operands and operators that reduces to a single value.

Ex: 20 + 20 = 40

An operand is a sub expression on which operator acts. In the above example 20, 20 are called as “operands”

Different types of operators are:

- 1) Unary operator
- 2) Binary operator
- 3) Ternary operator

1) Unary operator: - If any operator performs some operation on only one variable, such an operator can be called as unary operator.

Ex: ++,--

2) Binary operator: - If any operator performs some operation on two variables, such an operator can be called as binary operator.

3) Ternary operator: - If any operator performs some operation on more than two variables, such an operator can be called as Ternary operator.

Ex:?:

Every operator in "C", May be unary or binary or Ternary operator.

They include:

1. Arithmetic
2. Relational
3. Logical
4. Assignment
5. Increment and Decrement
6. Conditional
7. Bitwise
8. Special

Arithmetic Operators

The arithmetic operators perform arithmetic operations. Operator

Meaning

+	Addition or Unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulo division

Ex: a+b, a-b, a*b, a/b, a%b

Here a and b are variables and also known as operands. The division modulo operator % cannot be used on floating point data.

Integer Arithmetic:

If a=14 and b=4

a / b = 3

a % b = 2

Real Arithmetic:

If x, y=6.0 and z=7.0 are floats, then:

$x = y/z = 6.0/7.0 = 0.857143$

Mixed-mode Arithmetic:

If a = 25 and b = 10.0 then a/b = 2.5 where as 25/10 = 2

Addition of two numbers:

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

```
main() {
```

```
int a,b,c;
```

```
printf("enter two number's");
```

```
scanf("%d%d",&a,&b);
```

```
c=a+b;
```

```
printf("Additon : %d",c);
```

```
return 0;
}
Using arithmetic operators addition, subtraction, multiplication, division
#include<stdio.h>
main() {
int a,b,sum,add,sub,mul,div;
printf("enter a,b values");
scanf("%d%d",&a,&b);
sum=a+b;
sub=a-b;
mul=a*b;
div=a/b;
printf("%d%d%d%d",sum,sub,mul,div);
return 0;
}
```

(or)

```
#include <stdio.h>
int main(){
int a,b,c;
printf("enter a,b values");
scanf("%d%d",&a,&b);
c=a+b;
printf("a+b=%d\n",c);
c=a-b;
printf("a-b=%d\n",c);
c=a*b;
printf("a*b=%d\n",c);
c=a/b;
printf("a/b=%d\n",c);
c=a%b;
printf("Remainder when a divided by b=%d\n",c);
return 0;
}
}
```

Test your skill?

Output?

```
#include<stdio.h>
#include<conio.h>
int main() {
int a=10,b=5;
a = a-(-b);
printf("Sum is : %d" a);
return 0;
}
```


Output? # include
 <stdio.h>
 int main(){
 int x = 10;
 int y = 20;
 x += y += 10;
 printf (" %d %d", x, y);
 return 0;
 }

Relational Operators

The relational operators are used to compare two expressions or operands. An expression which contains relational operators such expression is called relational expression. The relational operators are:

Relational Operators	Operator	Meaning
<		Is less than
>		Is greater than
<=		is less than or equal to
>=		is greater than or equal to
==		Is equal to
!=		Is not equal to

Here ae-1, ae-2 are arithmetic expressions. The value of the relational expression is either one or zero. If the value is one then the specified expression is true otherwise false (value = 0).

Ex: 5.5 < 14 TRUE

20 > 27.5 FALSE

```
int main() {
int x = 20;
int y = 40;
printf("Value of %d > %d is %d\n",x,y,x>y);
printf("Value of %d >= %d is %d\n",x,y,x>=y);
printf("Value of %d <= %d is %d\n",x,y,x<=y);
printf("Value of %d < %d is %d\n",x,y,x<y);
return 0;
}
```

Logical Operators:

The logical operator is which combines two or more relational expressions into one expression. The Logical operators in 'C' are:

Symbol	Meaning
&&	logical AND
	logical OR
!	logical NOT

NOT – returns not false for true and not true for false.

Ex: i) age > 55 && salary < 1000

ii) number < 0 || number > 100

Assignment Operators

Assignment operators are used to assign the result of an expression to a variable. Assignment operator in 'C' is '='. The format of assignment operator is

Variable = expression;

Ex: a=10;

(or)

Variable op= expression;

Where 'op' is an arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment statement

v op= exp;

is equivalent to

v = v op (exp);

Ex: x += 2; is same as x = x + 2; //The statement is more efficient.

```
int main() {
    int a = 21;
    int c ;
    c = a;
    printf("Line 1 - = Operator Example, Value of c = %d\n", c );
    c += a;
    printf("Line 2 - += Operator Example, Value of c = %d\n", c );
    c -= a;
    printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
    c *= a;
    printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
    c /= a;
    printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
    return 0;
}
```

Increment and Decrement Operators

'C' has two very useful unary operators.

Increment operator (++) increments the variable by 1 and decrement operator (--) decrements the variable by 1. The increment and decrement operator can appear before or after the variable.

Ex: x++, ++x, x--, --x

If an Increment/Decrement operator is written before a variable then it is called Pre increment/decrement operator. And If an Increment/Decrement operator is written after a variable then it is called Post increment/decrement operator.

Test your skill:

Question:

What is the difference between ++i and i++?

++i will increment the value of i, and then return the incremented value.

i = 1;

j = ++i;

(i is 2, j is 2)

i++ will increment the value of i, but return the original value that i held before being incremented.

i = 1;

j = i++;

(i is 2, j is 1)

```
#include<stdio.h>
main()
{
const int num=12;
num++; //cannot modify the constant object
}
```

Output:

```
int main(){
int i=5,j;
j=++i+++i+++i;
printf("%d %d",i,j);
return 0;
}
```

Rule :- ++ is pre increment operator so in any arithmetic expression it first increment the value of variable by one in whole expression then starts assigning the final value of variable in the expression.

Compiler will treat this expression $j = ++i+++i+++i$; as

$i = ++i + ++i + ++i$;

Initial value of $i = 5$ due to three pre increment operator final value of $i=8$.

$j=8+8+8$

$j=24$ and $i=8$

Output:

```
int main(){
int i=1;
i=2+2*i++;
printf("%d",i);
return 0;
}
```

Explanation:

$i++$ i.e. when postfix increment operator is used any expression the it first assign the its value in the expression the it increments the value of variable by one. So,

$i = 2 + 2 * 1$

$i = 4$

Now i will be incremented by one so $i = 4 + 1 = 5$

Test your skill:

In the case of “ $x++$ ”, another way to write it is “ $x = x + 1$ ”

Output:

```
int main(){
int a=2,b=7,c=10;
c=a==b;
printf("%d",c);
return 0;
}
```

Explanation:

$==$ is relational operator which returns only two values.

0: If $a == b$ is false

1: If $a == b$ is true

Since

a=2

b=7

So, a == b is false hence b=0

Question:

What is the difference between the expression “++a” and “a++”?

In the first expression, the increment would happen first on variable a, and the resulting value will be the one to be used. This is also known as a prefix increment. In the second expression, the current value of variable a would be the one to be used in an operation, before the value of a itself is incremented. This is also known as postfix increment.

```
#include<stdio.h>
int main() {
int a=10;
int b=a++; // At line 4
```

```
printf("%d",b);
}
```

Output: 10

At line 4 if we replace with int b=++a; then it outputs value: 11

Conditional Operator

A ternary operator “?:” is available in ‘C’ to construct conditional expressions of the form Condition? True part: False part;

In this Condition is evaluated first. If the condition is true then the “True Statement” is evaluated otherwise “False Statement” is evaluated.

Ex: a = 10;

b = 15;

x = (a>b)?a:b; //Output: In this example, the value of ‘b’ will be assigned to x.

Bitwise Operators

‘C’ supports some special operators known as bitwise operators for manipulate of data at bit level. These operators may not be applied to float or double numbers. The bitwise operators are

Bitwise Operator	Meaning
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	One’s complement

```
int main() {
unsigned int a = 60; /* 60 = 0011 1100 */
unsigned int b = 13; /* 13 = 0000 1101 */
int c = 0;
c = a & b; /* 12 = 0000 1100 */
printf("Line 1 - Value of c is %d\n", c );
c = a | b; /* 61 = 0011 1101 */
printf("Line 2 - Value of c is %d\n", c );
```

```

c = a ^ b; /* 49 = 0011 0001 */
printf("Line 3 - Value of c is %d\n", c );
c = ~a; /* -61 = 1100 0011 */
printf("Line 4 - Value of c is %d\n", c );
c = a << 2; /* 240 = 1111 0000 */
printf("Line 5 - Value of c is %d\n", c );
c = a >> 2; /* 15 = 0000 1111 */
printf("Line 6 - Value of c is %d\n", c );
}

```

Question:

What does the C `??!` operator do?

It's a C trigraph:

`??!` being `|` so this is the operator `|`

Special Operators

'C' supports some special operators such as comma, size of, pointer operators (& and *) and member selection operator (. and - >).

The comma operator is used to separate a pair of expressions.

Ex: value = (x = 10, y = 5, x+y);

The sizeof is a compile time operator and to know the number of bytes the operand occupies.

sizeof() function in C:

sizeof() function is used to find the memory space allocated for each C data types.

```

#include <stdio.h>
#include <limits.h>
int main() {
int a;
char b;
float c;
double d;
printf("Storage size for int data type:%d \n",sizeof(a));
printf("Storage size for char data type:%d \n",sizeof(b));
printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
return 0;
}

```

Storage size for char data type:1

Storage size for float data type:4

Storage size for double data type:8

Test your skill: If *char*=1, *int*=4, and *float*=4 bytes size, What will be the output of the program ?

```

#include<stdio.h>
int main() {
char ch = 'A';
printf("%d, %d, %d", sizeof(ch), sizeof('A'), sizeof(3.14f));
return 0;
}

```

Answer: 1,4,4

Explanation:

Step 1: *char ch = 'A';* The variable *ch* is declared as an character type and initialized with value 'A'.

Step 2:

printf("%d, %d, %d", sizeof(ch), sizeof('A'), sizeof(3.14));

The sizeof function returns the size of the given expression.

sizeof(ch) becomes *sizeof(char)*. The size of *char* is 1 byte.

sizeof('A') becomes *sizeof(65)*. The size of *int* is 4 bytes (as mentioned in the question).

sizeof(3.14f). The size of *float* is 4 bytes.

Hence the output of the program is 1, 4, 4

Output:

Assume that a character takes 1 byte. Output of following program? `#include<stdio.h>`

```
int main() {
char abc[20] = "lkgtopg.in";
printf ("%d", sizeof(abc));
return 0;
}
```

Output: ?

Explanation: Note that the `sizeof()` operator would return size of array. To get size of string stored in array, we need to use `strlen()`. The following program prints __

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[20] = "lkgtopg.in";
printf ("%d", strlen(str));
return 0;
}
```

Output:

What will be output when you will execute following c code?

```
#include<stdio.h>
int main(){
double num=9.2;
int var=5;
printf("%d",sizeof(!num));
printf("%d",sizeof(var=25/3));
printf("%d",var); return 0;
}
```

Explanation:

Output : 2 2 5

`sizeof(Expr)` operator always returns the an integer value which represents the size of the final value of the expression `expr`.

Consider on the following expression:

!num
 !=5.2
 =0
 0 is int type integer constant and its size is 2
 Consider on the following expression:
 var = 25/3
 => var = 8
 => 8
 8 is int type integer constant.

Output:

```
#include<stdio.h>
int main(){
char a=250;
int expr;
expr= a+ !a + ~a + ++a;
printf("%d",expr); return 0;
}
```

char a = 250;

250 is beyond the range of signed char. Its corresponding cyclic value is: -6

So, a = -6

Consider on the expression:

expr= a+ !a + ~a + ++a;

Operator!, ~ and ++ have equal precedence. And its associativity is right to left.

So, First ++ operator will perform the operation. So value a will -5

Now,

Expr = -5 + !-5 + ~-5 + -5

= -5 + !-5 + 4 - 5

= -5 + 0 + 4 - 5

= -6

Output:

```
#include<stdio.h>
int main(){
int a=2,b=7,c=10;
c==b;
printf("%d",c);
return 0;
}
```

Output: 0

Explanation: == is relational operator which returns only two values.

0: If a == b is false

1: If a == b is true

Since

a=2

b=7

So, a == b is false hence b=0

Output:

```
#include<stdio.h>
void main(){
int x;
x=10,20,30;
printf("%d",x);
return 0;
}
```

Explanation:

Precedence table:

Operator	Precedence	Associative
=	More than ,	Right to left
,	Least	Left to right

Since assignment operator (=) has more precedence than comma operator. So = operator will be evaluated first than comma operator. In the following expression

$x = 10, 20, 30$

First 10 will be assigned to x then comma operator will be evaluated.

Output:

```
#include<stdio.h>
int main(){
int a=0,b=10;
if(a=0){
printf("true");
}
else{
printf("false");
}
return 0;
}
```

Explanation:

As we know = is assignment operator not relation operator. So, $a = 0$ means zero will assigned to variable a. In c zero represent false and any non-zero number represents true.

So, if(0) means condition is always false hence else part will execute.

Output:

```
#include<stdio.h>
int main(){
int a;
a=015 + 0x71 +5;
printf("%d",a);
return 0;
}
```

Explanation: 015 is octal number its decimal equivalent is $= 5 * 8^0 + 1 * 8^1 = 5 + 8 = 13$

0x71 is hexadecimal number (0x is symbol of hexadecimal) its decimal equivalent is $= 1 * 16^0 + 7 * 16^1 = 1 + 112 = 113$

So, $a = 13 + 113 + 5 = 131$

Output:

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

```
int main(){
int x=100,y=20,z=5;
printf("%d %d %d");
return 0;
}
```

Output: 5 20 100

By default x, y, z are auto type data which are stored in stack in memory. Stack is LIFO data structure. So in stack first stores 100 then 20 then 5 and program counter will point top stack i.e. 5. Default value of %d in printf is data which is present in stack. So output is reverse order of declaration. So output will be 5 20 100.

Output:

```
#include<stdio.h>
int main(){
int a;
a=sizeof(!5.6);
printf("%d",a);
return 0;
}
```

Output:

Explanation:

! is negation operator it return either integer 0 or 1. ! Any operand = 0 if operand is non zero. ! Any operand = 1 if operand is zero. So, !5.6 = 0
Since 0 is integer number and size of integer data type is two byte.

Output:

```
#include<stdio.h>
int main(){
int i=5;
int a=++i + ++i + ++i;
printf("%d",a);
return 0;
}
```

In the following expression:

```
int a=++i + ++i + ++i;
```

Here break point is due to declaration .It break after each increment i.e. (initial value of i=5) after first increment value 6 assign to variable i then in next increment will occur and so on.

So, a = 6 + 7 + 8;

```

7. printf (" %d welcome %d", 10, 20); // 10 welcome 20
8. printf (" %d %d %d", 10, 20, 30); // 10 20 30
9. printf (" %d %d %d", 10, 20, 30); // 10 20 30
10. printf (" %d, %d, %d", 10, 20, 30); // 10, 20, 30
11. printf (" 2+3=%d", 2+3); // 2+3=5
12. printf (" %d * %d = %d", 2, 3, 2*3); // 2 * 3 = 6

```

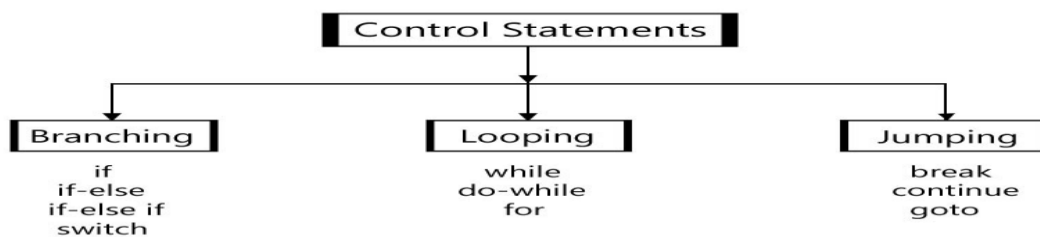
Control statements:

Alternatively referred to as flow of control, control flow (when referring to computer programming) is the order function calls, instructions, and statements are executed or evaluated when a program is running.

Flow control describes the order in which all the statements will execute at run time.

Flow controls are categorized into 3 types those are

- 1) Selection statements
- 2) Iterative statements
- 3) Transfer statements



Selection/Decision/Branching statements:

Selection statements are also called as decision making statements.

When we are working with selection statement, then if condition is true, then corresponding block will be executed, if condition is false, then the block will not be executed.

Syntax: if(condition) {

Statement 1;

Statement 2; //true

} //false

Ex: #include<stdio.h>

main(){

int a=5,b=5;

if((a-b)==0){

printf(" a and b are equal");

}

For a single statement it is not necessary to place body.

else: We can create alternate block of if condition. If we are using if-else at point of time only one block will be executed.

```
if(condition){  
    Stmt1;  
    Stmt2;  
}  
else {  
}
```

if-else statement

```
#include<stdio.h>  
int main(){  
    int a=20,b=10;  
    if(a>b){  
        printf("a is big");  
    }  
    else{  
        printf(" b is big");  
    }  
    return 0;  
}
```

switch statement:

In some programming situation, when there are number of choices and we want to choose only appropriate choice, in that situation C provided **switch** statement. Thus **switch** statement allows us to make a decision from the number of choices. The **switch** statement also known as **switch-case-default**. Since, the switch statement known for decision maker.

*Syntax of **switch** statement:* switch(*integer expression*){

```
case 1 : do this; break;  
case 2 : do this; break;  
case 3 : do this; break;  
case 4 : do this; break;  
default : and this;  
break;  
}
```

Example:

```
#include<stdio.h>  
main() {  
    int n =4;  
    switch(n) {  
        case1: printf("red");  
        break;  
        case2: printf("blue");  
        break;  
        case3: printf("orange");  
        break;  
        case4: printf("black");  
        break;  
    }
```

```
default: printf("no match");
}
return 0;
}
```

Without break:

example without break:

```
#include<stdio.h>
main() {
int n =1;
switch(n) {
case1:
printf("red");
case2:
printf("blue");
case3:
printf("orange");
case4:
printf("black");
default:
printf("no match");
}}
```

We can't use float value with switch statement because floats are imprecise and we never know what that number is actually going to be.

That is the reason we use character (char) , integer(int) . enumeration (enum) can also be used

```
char inchar = 'A';
```

```
switch (inchar)
```

```
{
```

```
case 'A' :
```

```
printf ("choice A \n") ;
```

```
case 'B' :
```

```
printf ("choice B ") ;
```

```
case 'C' :
```

```
case 'D' :
```

```
case 'E' :
```

```
default:
```

```
printf ("No Choice") ;
```

```
}
```

(A) No choice

(B) Choice A

(C) Choice A

Choice B No choice

(D) Program gives no output as it is erroneous

Answer (C)

There is no break statement in case 'A'. If a case is executed and it doesn't contain break, then all the subsequent cases are executed until a break statement is found.

Iterative/Loops:

Set of instructions given to the compiler to execute set of statements, until the condition becomes false it is called as loop.

Iteration statements are called loop. Because repetition will be in the form of circle. That's why iteration statements are called loops.

In C programming language, loops are classified into three types: while , for dowhile

while loop

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

```
main() {
```

```
int i;
```

```
while(i<10){
```

```
printf("%d",i);
```

```
i++;
```

```
}}
```

do while loop

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

```
main(){
```

```
int i;
```

```
do{
```

```
printf("%d",i);
```

```
i++;
```

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

```
}
```

for loop

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

```
main(){
```

```
for(int i=0;i<10;i++)
```

```
{
```

```
printf("%d",i);
```

```
}
```

```
}
```

Test your skill:

How many times the following code prints the string "hello".

```
for(i=1; i<=1000; i++);
```

```
printf("hello");
```

(A) 1 (B) 1000

(C) Zero (D) Syntax error

Ans: A

The "for" loop is terminated by a semicolon so the next statement is execute that is printing hello.

Output:

```
int main(){
```

```
int i=1;
```

```
for(;i<=4;i++){
```

```
printf("%d ",i);
```

```
}
```

```
return 0;
```

```
}
```

Output: 1 2 3 4

Question:**Can the curly brackets { } be used to enclose a single line of code?**

While curly brackets are mainly used to group several lines of codes, it will still work without error if you used it for a single line. Some programmers prefer this method as a way of organizing codes to make it look clearer, especially in conditional statements.

Common Mistake: (Misuse of Semicolon)

Another common mistake is to put a semicolon in a wrong place.

Consider the following code:

```
for(i =1; i<=10;i++);
```

The code is supposed to the integers from 1 to 10.

But what actually happens is that only the last value of i is printed.

Jumping or transfer statements:

When we writing programming code, we often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test. The

keyword **break** allows us to do this. When **break** is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an if. The keyword **break**, breaks the control only from the while in which it is placed.

break:

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

```
main(){
```

```
for(int i=0;i<10;i++){
```

```
printf("%d",i);
```

```
if(i==5)
```

```
break;
```

```
}}
```

continue:

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

```
main(){
```

```
for(int i=0;i<10;i++){
```

```
printf("%d",i);
```

```
if(i==5)
```

```
continue;
```

```
}}
```

Errors:

Error: The difference between expected result and actual result is error Bug: If that error comes at the time of development stage before production then it is bug. Defect: If that error comes after production then we say it is defect.

While writing c programs, errors (The difference between expected result and actual result is error) also known as bugs (If that error comes at the time of development stage before production then it is bug) in the world of programming may occur unwillingly which may prevent the program to compile and run correctly as per the expectation of the programmer.

If we don't detect them and correct them, they cause a program to produce wrong results. There are three types of errors — syntax, logical, and run-time errors. Let us look at them:

Syntax errors: These errors occur because of wrongly typed statements, which are not according to the syntax or grammatical rules of the language.

For example, in C, if you don't place a semi-colon after the statement (as shown below), it results in a syntax error.

- ✓ Missing semicolon (;) at the end of statement.
- ✓ Missing any of delimiters i.e { or }
- ✓ Incorrect spelling of any keyword.
- ✓ Using variable without declaration etc.

Logical errors: These errors occur because of logically incorrect instructions in the program. Let us assume that in a 1000 line program, if there should be an instruction, which multiplies two numbers and is wrongly written to perform addition. This logically incorrect instruction may produce wrong results.

Runtime errors: These errors occur during the execution of the programs though the program is free from syntax and logical errors. Some of the most common reasons for these errors are

- ✓ When you instruct your computer to divide a number by zero.
- ✓ When you instruct your computer to find algorithm of a negative number.
- ✓ When you instruct your computer to find the square root of a negative integer.
- ✓ Trying to open a file which is not created
- ✓ Lack of memory space

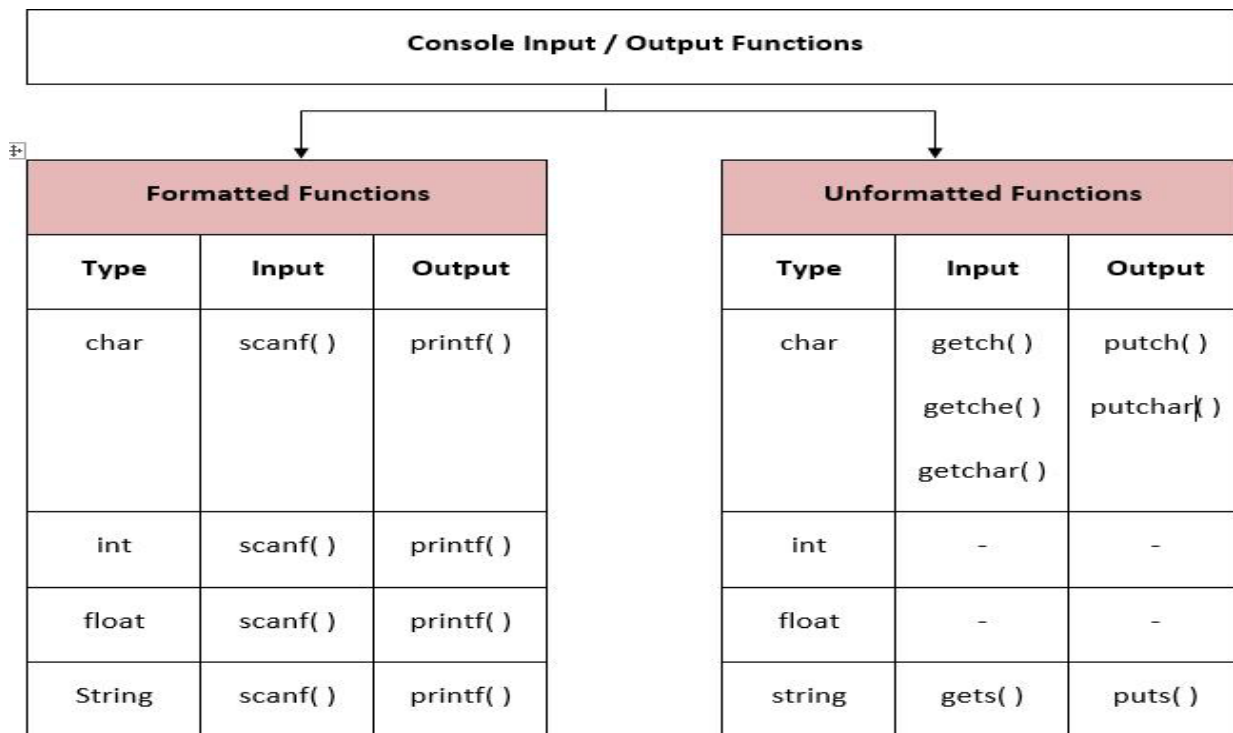
Generally occurs due to some illegal operation performed in the program.

Linking Error: These errors are occurs due to the linking of header files to the current program. Unless you run the program, there is no chance of detecting such errors

Common Mistake: Missing Braces

It is common to forget a closing brace when coding a deeply nested loop. The number of opening braces should match with the closing ones. However, if we put a matching brace in a wrong place, the compiler won't notice the mistake and the program will produce unexpected result.

Input and output functions:



scanf () :function is used to read values using key board. It is used for runtime assignment of variables.

The general form of scanf() is scanf("format String ", list_of_addresses_of_Variables);

The format string contains - Conversion specifications that begin with % sign

Eg: scanf(" %d %f %c", &a &b, &c) ;

"&" is called the "address" operator. In scanf() the "&" operator indicates the memory location of the variable. So that the Value read would be placed at that location.

printf() : function is used to Print / display values of variables using monitor:

The general form of printf() is printf("control String ", list_of_Variables);

- Characters that are simply printed as they are
- Conversion specifications that begin with a % sign
- Escape sequences that begin with a "\" sign.

getchar () function is used to read one character at a time from the key board

Syntax ch = getchar (); where ch is a char Var.

```
main ( ) {
char ch;
printf("Enter a char");
ch = getchar ( );
printf("ch =%c", ch);
}
```

O/P Enter a char M

ch = M

When this function is executed, the computer will wait for a key to be pressed and assigns the value to the variable when the "enter" key pressed.

putchar () : function is used to display one character at a time on the monitor.

Syntax: putchar (ch);

Ex: char ch = "M";

putchar (ch);

The Computer displays the value char of variable "ch" i.e. M on the Screen.

getch () : function is used to read a char from a key board and does not expect the "enter" key press.

Syntax: `ch = getch ();`

When this function is executed, computer waits for a key to be pressed from the key board. As soon as a key is pressed, the control is transferred to the next line of the program and the value is assigned to the char variable.

Function	Prototype	Header File	Description
<code>getchar()</code>	<code>int getchar(void);</code>	<code>stdio.h</code>	Returns from standard input device, namely keyboard
<code>gets()</code>	<code>char *gets(char *string);</code>	<code>stdio.h</code>	Returns a string from the keyboard
<code>getch()</code>	<code>int getch(void);</code>	<code>conio.h</code>	Gets a character from the keyboard without echoing, that is, it does not appear on the screen
<code>getche()</code>	<code>int getche(void);</code>	<code>conio.h</code>	Gets a character from the keyboard and echoes on the screen
<code>scanf()</code>	<code>scanf(const char * str, arg);</code>	<code>stdio.h</code>	Gets different type of data depending on the conversion characters in the constant string

getche (): function is used to read a char from the key board without expecting the enter key to be pressed. The char read will be displayed on the monitor.

Syntax: `ch = getche ();`

Note that `getche ()` is similar to `getch ()` except that ***getche () displays the key pressed from the key board on the monitor. In getch () “e” stands for echo.***

String I/O functions

`gets ()` function is used to read a string of characters including white spaces. Note that white spaces in a string cannot be read using `scanf ()` with `%s` format specifier.

Syntax: `gets (S);` where “S” is a char string variable

Ex: `char S[20];`

`gets (S);`

Function	Prototype	Header File	Description
<code>putchar()</code>	<code>int putchar(int ch);</code>	<code>stdio.h</code>	Puts the character on the screen
<code>puts()</code>	<code>int puts(char string);</code>	<code>stdio.h</code>	Puts the string on the screen
<code>putch()</code>	<code>int putch(int ch);</code>	<code>conio.h</code>	Puts the character on the screen
<code>printf()</code>	<code>printf(const char *str, arg);</code>	<code>stdio.h</code>	Puts the constant string and the arguments on the screen depending on the conversion characters in the constant string

Common Mistake: Missing & operator in scanf Parameters

All non-pointer variables in a `scanf` call should be preceded by an & operator.

If the variable code an integer, then the statement

`scanf(“%d”, a);` is wrong

The correct one is `scanf(“%d”, &a);`

Remember that the compiler will not detect the error and you may get a crazy output.

Test your skill:

```
#include<stdio.h>
int main()
{
int a=5,b=6,c=11;
clrscr();
printf("%d %d %d");
getch();
}
```

(a)Garbage value garbage value garbage value

(b)5 6 11

(c)11 6 5

(d)Compiler error

Answer: (c)

Output:

What is wrong in this statement? `scanf("%d",num);`

An ampersand & symbol must be placed before the variable name num. Placing & means whatever integer value is entered by the user is stored at the “address” of the variable name. This is a common mistake for programmers, often leading to logical errors.

Arrays

An array is collection of same data type elements in a single entity. Or an array is collection of homogeneous elements in a single variable i.e., you can store group of data of same data type in an array.

It allocates sequential memory locations, Individual values are called as elements. Always, Contiguous (adjacent) memory locations are used to store array elements in memory. It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Example for C Arrays:

```
int a[10]; // integer array
```

```
char b[10]; // character array i.e. string
```

Types of Arrays: We can use arrays to represent not only simple lists of values but also tables of data in two or three or more dimensions.

One – dimensional arrays

Two – dimensional arrays

Multidimensional arrays

ONE – DIMENSIONAL ARRAY: A list of items can be given one variable name using only one subscript and such a variable is called a single – subscripted variable or a one – dimensional array.

Declaration of One-Dimensional Arrays :

Like any other variables, arrays must be declared before they are used.

The general form of array declaration is

Syntax: `<datatype> <array_name>[sizeofarray];`

The data type specifies the type of element that will be contained in the array, such as int, float, or char.

The size indicates the maximum number of elements that can be stored inside the array.

The size of array should be a constant value.

`float height[50];` // Declares the height to be an array containing 50 real elements. Any subscripts 0 to 49 are valid.

`int group[10];` // Declares the group as an array to contain a maximum of 10 integer constants.

Test your skill for Valid Statements:

`a = number[0] + 10;`

`number[4] = number[0] + number[2];`

`number[2] = x[5] + y[10];`

`value[6] = number[i] * 3;`

One dimensional array in C:

Syntax : `data-type arr_name[array_size];`

Array declaration	Array initialization
Syntax: <code>data_type arr_name [arr_size];</code>	<code>data_type arr_name [arr_size]=(value1, value2, value3,...);</code>
<code>int age [5];</code>	<code>int age[5]={0, 1, 2, 3, 4};</code>
<code>char str[10];</code>	<code>char str[10]='H','a','i';</code> (or) <code>char str[0] = 'H';char str[1] = 'a';char str[2] = 'i';</code>

Two dimensional arrays in C:

Two dimensional array is nothing but array of array.

syntax : `data_type array_name[num_of_rows][num_of_column]`

S.no	Array declaration	Array initialization	Accessing array
1	Syntax: <code>data_type arr_name [num_of_rows][num_of_column];</code>	<code>data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};</code>	<code>arr_name[index];</code>
2	Example: <code>int arr[2][2];</code>	<code>int arr[2][2] = {1,2, 3, 4};</code>	<code>arr [0] [0] = 1; arr [0] [1] = 2;arr [1][0] = 3; arr [1] [1] = 4;</code>

Test your skill?

To declare an array S that holds a 5-character string, you would write

(A) `char S[5]` (B) `String S[5]`

(C) `char S[6]` (D) `String S[6]`

Ans: A

A string is nothing but a char array.

Why is it necessary to give the size of an array in an array declaration?

Ans: When an array is declared, the compiler allocates a base address and reserves enough space in memory for all the elements of the array. The size is required to allocate the required space and hence size must be mentioned.

Output:

`#include<stdio.h>`

```
void main(){
char arr[6]="Gsstec";
printf("%s",arr);
}
```

Answer: ????

Output:

```
#include<stdio.h>
void main(){
char arr[11]="The African";
printf("%s",arr);
}
```

Output: The African

Output:

```
#include<stdio.h>
void main(){
char arr[11]="The African queen";
printf("%s",arr);
}
```

Output: compilation Error

Output:

```
sizeofArray:
#include<stdio.h>
void main(){
char arr[20]="MysticRiver";
printf("%d",sizeof(arr));
}
```

Output: ??

Output:

The output of the following statements is

```
char ch[6]={'e', 'n', 'd', '\0', 'p'};
printf("%s", ch);
```

(A) endp (B) endOp

(C) end (D) error

Ans: C

printf statement will print end because string is terminated at "\0" and in array after d, we have null character.

Output:

```
int a[4];
a[0]=1;
a[1]=2;
a[2]=3;
a[3]=4;
Then : a[a[0+2]] = 4
```

Functions

Functions are subprograms which are used to compute a value or perform a task. They cannot be run independently and are always called by the main () function or by some other function. There are two kinds of functions

1. **Library or built-in functions** are used to perform standard operations eg: squareroot of a number `sqrt(x)`, absolute value `fabs(x)`, `scanf()`, `printf()`, and so on. These functions are available along with the compiler and are used along with the required header files such as `math.h`, `stdio. h`, `string.h` and so on at the beginning of the program.
2. **User defined functions** are self-contained blocks of statements which are written by the user to compute a value or to perform a task. They can be called by the `main()` function repeatedly as per the requirement.

What are built in functions?

Ans: The functions that are predefined and supplied along with the compiler are known as Builtin functions. They are also known as library functions.

```
#include <stdio.h>
#include <conio.h>
int add(int p,int q);
int main()
{
    int a,b,c;
    clrscr();
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    c= 30;
    printf("\nSum of %d and %d is %d",a,b,c);
    getch();
    return 0;
}
int add(int p,int q)
{
    int result;
    result=p+q;
    return(result);
}
```

Enter two numbers
10
20

Memory diagram:
a: 10
b: 20
c: 30
p: 10
q: 20
result: 30

```
#include <stdio.h>
void display(int a)
{
    printf("%d",a);
}
int main(){
    int c[]={2,3,4};
    display(c[2]); //Passing array element c[2] only.
    return 0;
}
```

Output : 2

Pointers

Pointers are used in C program to access the memory and manipulate the address. C Pointer is a variable that stores/points the address of another variable. C Pointer is used to allocate

memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Syntax : data_type *var_name; Example : int *p; char *p;

Where, * is used to denote that "p" is pointer variable and not a normal variable.

Reference operator (&)

If var is a variable then &var is the address in memory. & symbol is used to get the address of the variable.

Normal variable stores the value whereas pointer variable stores the address of the variable.

The content of the C pointer always be a whole number i.e. address.

Always C pointer is initialized to null, i.e. int *p = null.

The value of null pointer is 0.

* symbol is used to get the value of the variable that the pointer is pointing to.

If pointer is assigned to NULL, it means it is pointing to nothing.

Two pointers can be subtracted to know how many elements are available between these two pointers.

But, Pointer addition, multiplication, division are not allowed.

The size of any pointer is 2 byte (for 16 bit compiler).

Example program for pointer in C:

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

```
int main()
```

```
{
```

```
int *ptr, q;
```

```
q = 50;
```

```
/* address of q is assigned to ptr */
```

```
ptr = &q;
```

```
/* display q's value using ptr variable */
```

```
printf("%d", *ptr);
```

```
return 0;
```

```
}
```

output :

50

```
/* Example to demonstrate use of reference operator in C programming. */
```

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

```
int main(){
```

```
int var=5;
```

```
printf("Value: %d\n",var);
```

```
printf("Address: %d",&var); //Notice, the ampersand(&) before var.
```

```
return 0;
```

```
}
```

Output

Value: 5

Address: 2686778

Test your skill?

1- Array of pointers, e.g , int *a[10]; Array of pointers to integer

- 2-Pointers to an array,e.g , int (*a)[10]; Pointer to an array of into
- 3-Function returning a pointer,e.g, float *f() ; Function returning a pointer to float
- 4-Pointer to a pointer ,e.g, int **x; Pointer to a pointer to int
- 5-pointer to a data type ,e.g, char *p; pointer to char

Are the expressions *ptr ++ and ++ *ptr same?

Ans: No, *ptr ++ increments pointer and not the value pointed by it. Whereas ++ *ptr increments the value being pointed to by ptr.

Differentiate between a constant pointer and pointer to a constant?

Ans:

const char *p; //pointer to a const character.

char const *p; //pointer to a const character.

char * const p; //const pointer to a char variable.

const char * const p; // const pointer to a const character.

What is an array of pointers?

Ans: if the elements of an array are addresses, such an array is called an array of pointers

What is pointer to a pointer?

Ans: If a pointer variable points another pointer value. Such a situation is known as a pointer to a pointer.

Example:

```
int *p1,**p2,v=10;
```

```
p1=&v; p2=&p1;
```

Here p2 is a pointer to a pointer

Strings

C Strings are nothing but array of characters ended with null character ('\0').

This null character indicates the end of the string.

Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes in C.

In C programming, array of character are called strings. A string is terminated by null character /0. For example:

```
"lkgtopg.in"
```

Here, "lkgtopg.in" is a string. When, compiler encounters strings, it appends null character at the end of string.

Declaration of strings

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of char type.

```
char s[5];
```

Initialization of strings

In C, string can be initialized in different number of ways.

```
char c[]="abcd";
```

OR,

```
char c[5]="abcd";
```

OR,

```
char c[]={ 'a','b','c','d','\0'};
```

OR;

```
char c[5]={ 'a','b','c','d','\0'};
```

Write a C program to illustrate how to read string from terminal.

```
#include <stdio.h>
int main(){
char name[20];
printf("Enter name: ");
scanf("%s",name);
printf("Your name is %s.",name);
return 0;
}
```

Enter name: Dennis
Your name is Dennis.

S.no	String functions	Description
1	<u>strcat ()</u>	Concatenates str2 at the end of str1.
2	<u>strncat ()</u>	appends a portion of string to another
3	<u>strcpy ()</u>	Copies str2 into str1
4	<u>strncpy ()</u>	copies given number of characters of one string to another
5	<u>strlen ()</u>	gives the length of str1.
6	<u>strcmp ()</u>	Returns 0 if str1 is same as str2. Returns <0 if str1 < str2. Returns >0 if str1 > str2.
7	<u>strcmpi (.)</u>	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
8	<u>strchr ()</u>	Returns pointer to first occurrence of char in str1.
9	<u>strrchr ()</u>	last occurrence of given character in a string is found
10	<u>strstr ()</u>	Returns pointer to first occurrence of str2 in str1.
11	<u>strrstr ()</u>	Returns pointer to last occurrence of str2 in str1.
12	<u>strdup ()</u>	duplicates the string
13	<u>strlwr ()</u>	converts string to lowercase
14	<u>strupr ()</u>	converts string to uppercase
15	<u>strrev ()</u>	reverses the given string
16	<u>strset ()</u>	sets all character in a string to given character
17	<u>strnset ()</u>	It sets the portion of characters in a string to given character

Include the corresponding header file "#include<string.h>

code : strlen()

#include <stdio.h>

#include <conio.h>

```

int main()
{
char str[30]="lkgtopg.in";
clrscr();
printf("\n\nstr : %s\n\n",str);
printf("\nlength of the string, strlen(str) is %d\n",strlen(str));
getch();
return 0;
}

```

Test your skill:

```

#include<stdio.h>
#include<string.h>
int main()
{
printf("%d\n", strlen("7997085322"));
return 0;
}

```

Explanation:

The function strlen returns the number of characters in the given string.

Therefore, strlen("7997085322") returns 10.

Hence the output of the program is "10".

code : strlwr()

```

#include <stdio.h>
#include <conio.h>
int main()
{
char str[30]="lkgtopg.in";
clrscr();
printf("\n\nstr : %s\n\n",str);
printf("strlwr(str) : %s\n",strlwr(str));
getch();
return 0;
}

```

code :strupr()

```

#include <stdio.h>
#include <conio.h>
int main()
{
char str[30]="lkgtopg.in";
clrscr();
printf("\n\nstr : %s\n\n",str);
printf("strupr(str) : %s\n",strupr(str));
getch();
return 0;
}

```

```
code: strcat()
#include <stdio.h>
#include <conio.h>
int main()
{
char str1[25]="i like ";
char str2[15]="lkgtopg.in";
clrscr();
printf("\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
printf("\n\nstrcat(str1,str2) : %s\n\n",strcat(str1,str2));
printf("\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
getch();
return 0;
}
```

```
code : strncat()
#include <stdio.h>
#include <conio.h>
int main()
{
char str1[25]="i like ";
char str2[13]="lkgtopg.in";
clrscr();
printf("\n\nbefore:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
printf("\n\nstrncat(str1,str2,5) : %s\n\n",strncat(str1,str2,5));
printf("\n\nafter:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
getch();
return 0;
}
```

```
code: strcpy()
#include <stdio.h>
#include <conio.h>
int main()
{
char str1[15]="i like ";
char str2[15]="lkgtopg.in";
clrscr();
printf("\n\nbefore:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
strcpy(str1,str2);
printf("\n\nafter:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
getch();
return 0;
}
```

```

code: strncpy()
#include <stdio.h>
#include <conio.h>
int main()
{
char str1[25]="i like ";
char str2[15]="lkgtopg.in ";
clrscr();
printf("\n\nbefore:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
strncpy(str1,str2,4);
printf("\n\nafter:\n\nstr1 : %s\t\t\tstr2 : %s\n\n",str1,str2);
getch();
return 0;
}

```

```

code: strcmp()
#include <stdio.h>
#include <conio.h>
int main() {
char str1[]="i like";
char str2[]="Hello";
int i;
clrscr();
i=strcmp(str1,str2);
if(i==0)
printf("\n\nstr1 and str2 are identical");
else if(i<0)
printf("\n\nstr1< str2");
else
printf("\n\nstr1< str2");
getch();
return 0;
}

```

Question:**Can the “if” function be used in comparing strings?**

No. “if” command can only be used to compare numerical values and single character values. For comparing string values, there is another function called strcmp that deals specifically with strings.

gets() and puts()

Functions gets() and puts() are two string functions to take string input from user and display string respectively as mentioned in previous chapter.

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

```

int main(){
char name[30];
printf("Enter name: ");
gets(name); //Function to read string from user.

```

```

printf("Name: ");
puts(name); //Function to display string.
return 0;
}
Using strlen() Function
#include <stdio.h>
int main() {
char s[1000],i;
printf("Enter a string: ");
scanf("%s",s);
for(i=0; s[i]!='\0'; ++i);
printf("Length of string: %d",i);
return 0;
}

```

Output:

Enter a string: lkgtopg

Length of string: 7

Program to print prefix and suffix of a string?

```

printf("enter a single character");
c=getchar();
printf("prefix of character %c",c);
putchar(c-1);
printf("suffix of character %c",c);
putchar(c+1);
}

```

```

#include <stdio.h>
int fun(){
puts(" Hello ");
return 10;
}
int main(){
printf("%d", sizeof(fun()));
return 0;
}

```

(2) Visibility of auto variable is within the block where it has declared. For examples:

(a) #include<stdio.h>

```

int main(){
int a=10;
{
int a=20;
printf("%d",a);
}
printf(" %d",a);
}

```



```
return 0;
}
```

Output: 20 10

Explanation: Visibility of variable a which has declared inside inner has block only within that block.

(b)

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

```
int main(){
```

```
{
```

```
int a=20;
```

```
printf("%d",a);
```

```
}
```

```
printf(" %d",a); //a is not visible here
```

```
return 0;
```

```
}
```

Output: Compilation error

Explanation: Visibility of variable which has declared inside inner block has only within that block.

Write a C program to print "lkgtopg.in" without using a semicolon

Use printf statement inside the if condition

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

```
int main()
```

```
{
```

```
if( printf( "lkgtopg.in" ) )
```

```
{ }
```

```
}
```

Write a C program to print ";" without using a semicolon

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

```
int main()
```

```
{
```

```
if(printf("%c",59))
```

```
{
```

```
}
```

```
}
```

```
int add(int x, int y)
```

```
{
```

```
return printf("%*c%*c", x, '\r', y, '\r');
```

```
}
```

```
int main()
```

```
{
```

```
printf("Sum = %d", add(3, 4));
```

```
return 0;
```

```
}
```

Output:

Sum = 7

Example:

```
#include<stdio.h>
int add(int, int); /* Function prototype */
int main()
{
    int a = 4, b = 3, c;
    c = add(a, b);
    printf("c = %d\n", c);
    return 0;
}
int add(int a, int b)
{
    /* returns the value and control back to main() function */
    return (a+b);
}
```

Output: c = 7

Library Name	Description
assert.h	Is to provide a definition of the assert macro, which prints an error message and aborts the program
alloc.h	There are functions to allocate, free memory, or obtain information from the memory blocks.
ctype.h	The functions that allow us to know the nature of a character, or to convert uppercase to lowercase and vice versa and integer values to ASCII codes.
dir.h	This lets you sort, create, modify, move and delete directories
errno.h	Represent the numbers of error, an error occurs then you can query the value of the system variable deerrno for more information about this error.
float.h	Define the limits of the floating-point types
limits.h	Define the boundaries of different types of integers
math.h	Contains the standard math functions used in C and C ++
setjmp.h	Defines the type of jmp_buf for some functions.
signal.h	Contains state functions.
stdarg.h	Defines functions that can be called with different numbers of arguments, so that they can write f(a) f(a, b).
stddef.h	Define some special types
stdio.h	Incorporate functions - Out E / S standard, types and macros
stdlib.h	Declare functions that are useful for different purposes, especially searching and sorting.
string.h	This file contains functions for handling strings.
time.h	Contains functions related to dates and times

Test your skills on these programs for given notes:

1. Write a Program to print "One Stop to your Search"?
2. Write a Program to accept declare a variable and initialize it?
3. Write a Program to accept an integer by user?
4. Write a program to declare variables with different names and data types?
5. Write a program to print remainder and quotient?

6. Write a program to accept five subject marks, total it and find average of total?
7. Write a program to perform addition of two numbers?
8. Write a program to perform addition, multiplication, subtraction, division of two Numbers?
9. Write a program to find reciprocal of a number?
10. Write a program to find Simple Interest?
11. Write a program to find the area of square?
12. Write a program to solve the equation $5x+3=2y$?
13. Write a program to perform sizeof operator?
14. Write a program to accept two variables and interchange it?
15. Write a program to find even or odd number of a given number?
16. Write a program to find whether a character is vowel or not?
17. Write a program to print 2 table?
18. Write a program to check whether a given number is multiple of 5 or not?
19. Write a program to perform any data type conversion?
20. Write a program to perform if-else condition?
21. Write a program to create calculator using switch case?
22. Write a program to check whether a given character is vowel or not?
23. Write a program to perform while loop, do-while loop, for loop?
24. Write a program to perform break and continue jumping statements?
25. Write a program to print first 1 to 10 natural numbers?
26. Write a program to print values from 10 to 1 reverse order?
27. Write a program to perform go-to statement?
28. Write a program to perform arithmetic operators?
29. Write a program to perform assignment operator?
30. Write a program to perform conditional/ternary operator?
31. Write a C program to check whether a number is even or odd using conditional/ternary Operator?
32. Write a program to perform increment and decrement of variable?
33. Write a program to check whether given number is +ve or -ve?
34. Write a program to demonstrate simple example of function?
35. Write a program to perform addition of two numbers using function?
36. Write a program to declare and initialize array and display array elements?
37. Write a program to print particular index array element?
38. Write a program to accept a character and String?
39. Write a program to perform putchar() and getchar()?
40. Write a program to perform gets and puts?
41. Write a program to find length of string?
42. Write a program to get prefix and suffix of a string?
43. Write a program to find ASCII value of 'A'?
44. Write a program to declare a pointer?
45. Write a program to access value from address using pointer?
46. Write a program to declare and print an array?

Basic Commonly Asked Programming Questions

- Write a program to perform addition without using addition operator?
- Find out whether a given number is perfect or not?
- Find out whether a given number is Armstrong or not?
- Find out whether a given number is Prime or not?
- Reverse of a number using c program?

Find out whether a given number is Strong number or not?
Write a program to find sum of digits of a number?
Write a program to check whether a given number is palindrome or not?
Find the Factorial of number using recursion and without using recursion?
Write a program to print lkgtopg.in without using semicolon?
Write a program to print Fibonacci series?
Write a program to check whether given string is palindrome?
Write a program to find sizes of data type without using sizeof operator?
Write a program to print a semicolon without using a semicolon anywhere in the code?

Hi guys we are back with a new start.

Lkgtopg.in Opportunity for any Graduate

A passion for technology, no matter what type - web, mobile, front end UI, back-end, marketing who has Good Working knowledge in PHP, Android any technology.

Enthusiastic individuals with creative and out of the box thinking in any technology. The journey might not be easy but will surely be rewarding. We have nothing to offer you right now in specific except for a dream which shall definitely come true with a collective effort and we can work on this as we build and grow our relationship. We have investors available to take care of the logistics part.

For Interns (Any graduate) if you're highly curious to learn new things and explore, we are ready provide you training so we don't charge you anything. Even our advisory is completely free of charge. If you're interested in any opportunities & curious to learn more about what we do and how we do it, please feel free to reach out to me at lkgtopgg@gmail.com or send your resumes to contact@lkgtopg.in
Want to help us or feedback, just drop a mail contact@lkgtopg.in