Mc Graw Hill

Introduction to Algorithms

19-09-23

* Representation of Algorithm :-

Defn:- A collection of finite number of sequential steps, when executed, that performs a given task is known as algorithm.

* Significance of Algorithm :-

→ It is very much useful for writing computer programs.
→ It acts as a blue print for writing programs.

* Characteristics of an Algorithm's (Properties) :-

1. Input : Zero or more quantities are externally supplied.
2. Output : At least one quantity is produced.
3. Definiteness : Each instruction is clear and unambiguous
4. finiteness : Algorithm should terminate after finite number of steps when traced in all cases.
5. Effectiveness : Every instruction must be basic. i.e, it can be carried out, by a person using pencil and paper

* Types of Algorithms :-

There are 3 types :-

1) Sequential.
2) Selection - check whether
3) Iterative - Repeat - until

* Sequential Ex: Write an algorithm to add 2 numbers and display in C.

Step 1:- Start
step 2: read 2 values a, b.
Step 3: add a, b and store in c
step 4: display C
Step 5: Stop.

* Write an Algorithm to find the sum and average of three numbers.

Step 1: Start
step 2: read 3 values a, b, c
Step 3: add a, b, c and store in d
Step 4: divide d by 3 and store in e
Step 5: display d, e
step 6: stop.

* write an algorithm to check the largest no

* Step 1: Start
step 2: read 2 values a, b.
step 3: check whether a is greater than b, if so go to step 4 otherwise go to step 5
Step 4: display a is largest no.
Step 5: display b is largest no.
Step 6: Stop.

* Merits and Demerits of Algorithm.

Merits:
→ Acts as a Blueprint for program development
→ Easy technique to understand logic
→ Easy to identify errors.
→ Easy to debug

DeMerits:
→ Time consuming
→ Difficult to code for lengthy problems
→ Big tasks are difficult to write in algorithms

21-9-23

* FlowChart :-

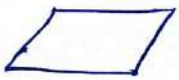
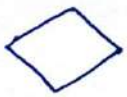
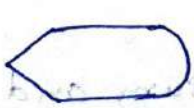
⇒ A flowchart is a visual representation of the sequence of steps followed for solving a problem

(or)

The diagrammatic or graphical or pictorial representation of the way in which to solve a given problem is called flow chart.

* Significance of Flowchart :-

→ Flow chart is an important aid in the development of a solution to a problem.
→ Easier to understand than an algorithm
→ Independent of any particular programming language
→ Proper documentation.
→ Proper debugging.
→ Easy and clear presentation

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning (or) end of the program.
Parallelogram		Denotes an input operation.
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of the two routes (if/then/else)
Hybrid		Denotes an output operation.
Flow line		Denotes the direction of logic flow in the program.

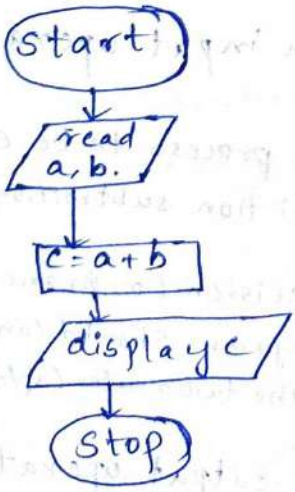
* Merits of Flow Chart :-

⇒ Short and simple
⇒ logical steps
⇒ Effective communication
⇒ Flowcharts can be visually appealing.
⇒ Visual clarity
⇒ Proper documentation

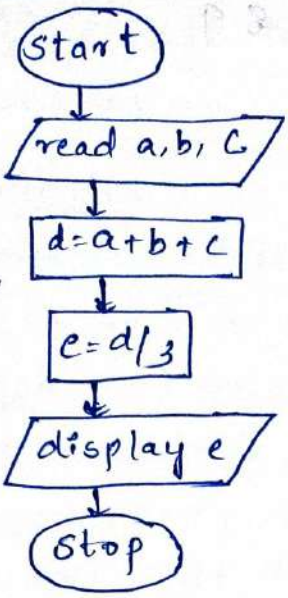
* De Merits :-

⇒ Not suitable where solution is long.
⇒ Complicate things
⇒ Difficult to alter
⇒ It can be difficult to draw a flowchart neatly, especially when mistakes are made.
⇒ Time taking
⇒ Including oversimplifying, repeat modifications and reproduction.

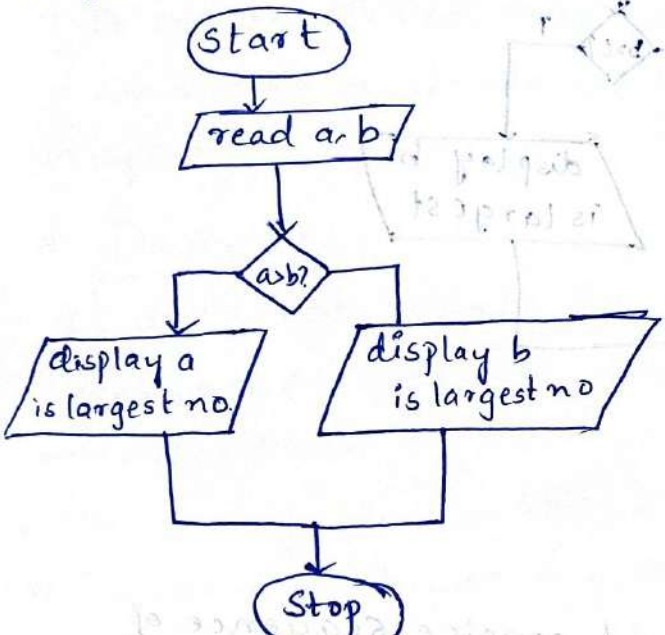
* Draw a Flowchart to add 2 numbers



* Draw a flowchart to display sum and average of 3 numbers.



* Draw a flowchart to find the largest of 2 numbers.



* write an algorithm and draw flow chart to a find. a largest of three no's.

Algorithm:-

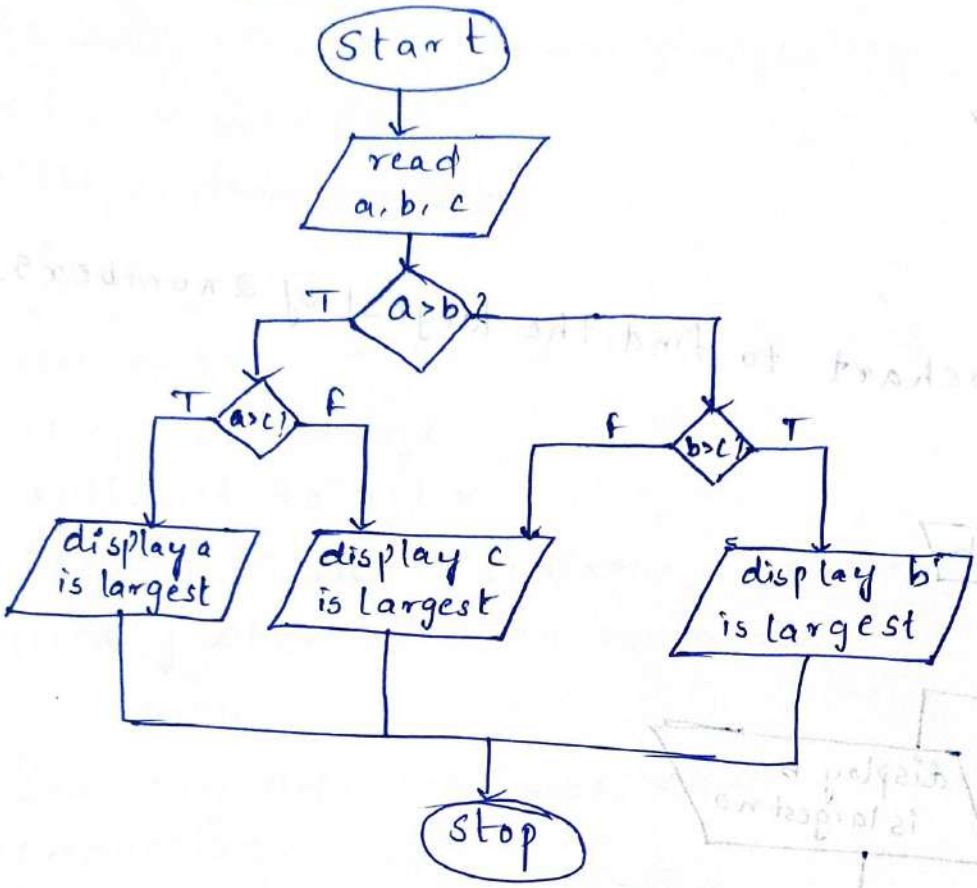
Step 1: Start
step 2 - read a, b, c values
Step 3 - check whether a is greater than b if so go to step 4 else go to step 6.
Step 4 - check whether a is greater than c if so go to step 5 else go to step 8.
Step 5 - display a is largest and go to step 9
Step 6 - check whether b is greater than c if so go to step 7 else go to step 8

Step 7- display b is largest and go to step 8 9

Step- 8 - display c is largest

Step 9 - Stop.

Flowchart



* Pseudocode :-

⇒ Pseudo code is a simple and concise sequence of English-like instructions to solve a problem.

⇒ Pseudo code is a "text-based" detail (algorithmic) design tool.

⇒ Pseudocode is often used as a way of describing a computer program to someone who doesn't understand how to program a computer.

* Significance :-

⇒ Pseudo code fits more easily on a page of Paper

⇒ Pseudocode can be written in a way that is very close to real program code

⇒ so making it easier later to write the program

* Pseudocode takes less time to write than drawing a flowchart.

* Merits :-

→ Improve the readability of any approach.
→ Acts as a bridge between the program and the algorithm or flowchart
→ It makes the code construction phase easier for the Programmer.
→ Pseudo code does not tend to run over many pages.
→ It can be easily modified as compared to flow chart.
→ It can be written, read and understood easily.
→ converting a pseudocode to programming language is very easy.

* DeMerits :-

→ It does not include the full logic of the proposed code
→ Pseudocode sometimes causes non programmers to misunderstand the complexity of a coding project.
→ The lack of standards is probably the main disadvantag of Pseudocode. Pseudocode is by nature unstructured; so the reader may not be able to see the logic in a step.
→ It's not visual
→ Introduce error possibilities in translating to coa

Ex. Pseudo code :- To determine whether student passe or failed.

```
start program
Input grade.
If student's grade is greater than or equal to 60
Print "Passed"
else
Print "failed"
ENDIF
End program.
```

Pseudocode: To compute the area of rectangle

Start program
Get the length, l and width, w
Compute the area = l*w
Display the area
End Program.

23-09-23

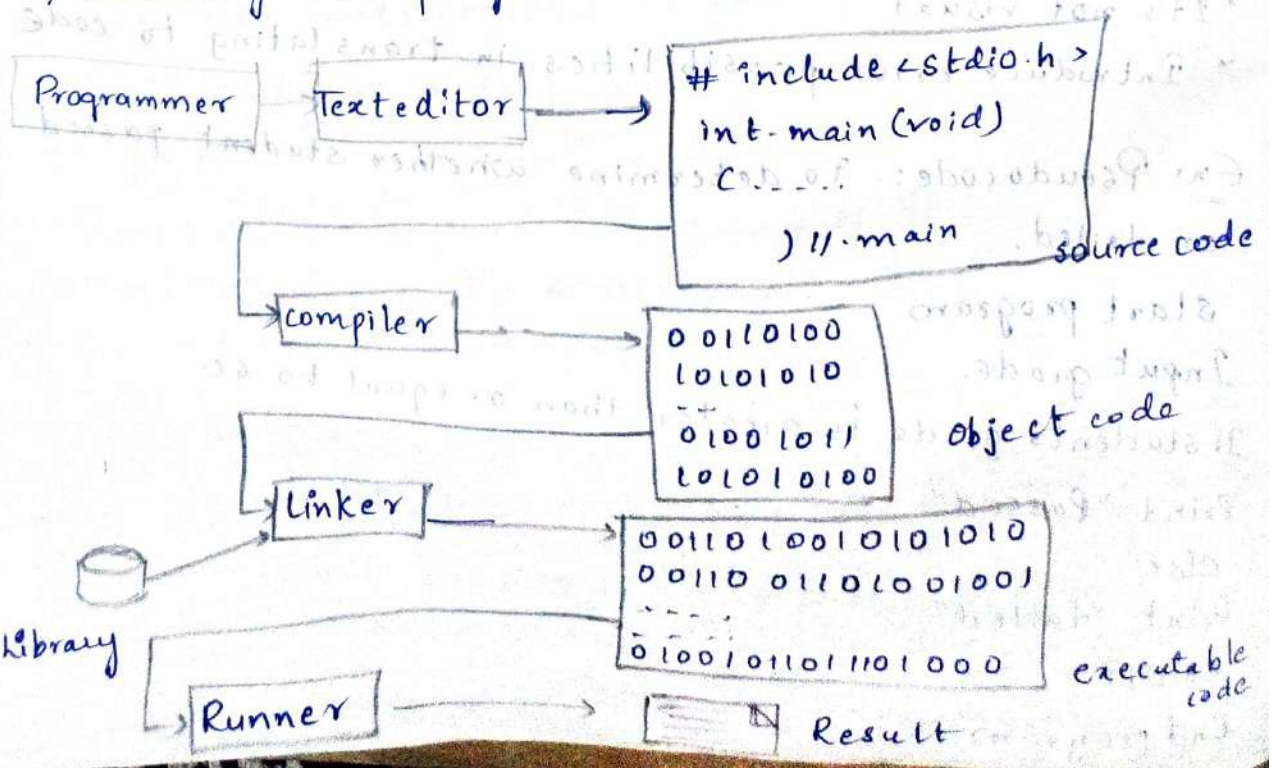
* Compiling and executing programmes:-

⇒ These are also called program development steps (PDS)

⇒ It is the job of programmer to write and test the Program.

⇒ The following are four steps for creating and running Programs:

1) Writing and editing the program
2) Compiling the program
3) Linking the program with the required library modules
4) Executing the program



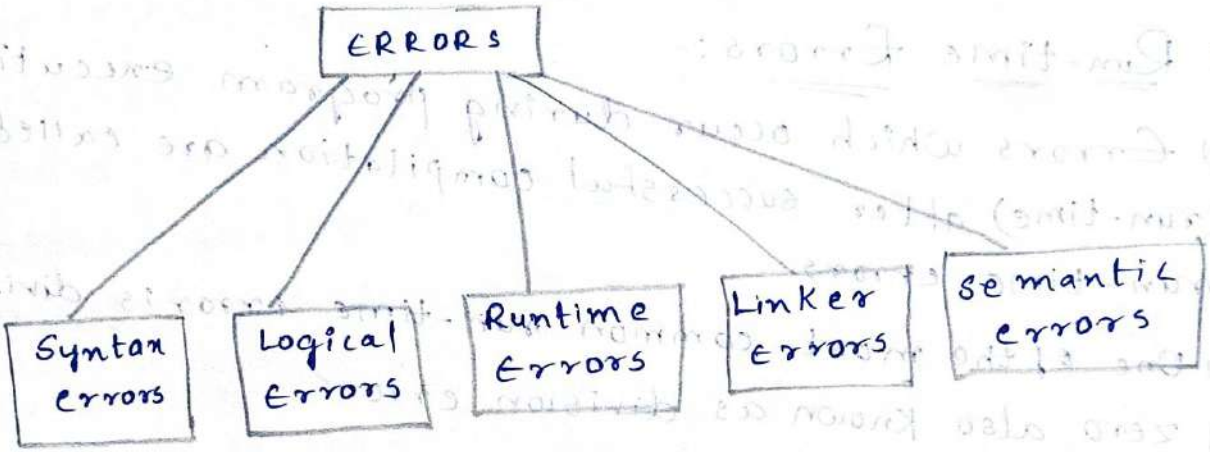
* Syntax and Logical errors:-

* ERROR:-

⇒ Error is a situation that causes a program not to run or halt its execution or give abnormal output

⇒ Error is an illegal operation performed by the user which results in abnormal working of the program.

* Types of errors in C



* Syntax errors:-

⇒ Errors that occur when you violate the rules of writing c/c++ syntax are known as syntax errors.

⇒ This compiler error indicates something that must be fixed before the code can be compiled

⇒ All these errors are detected by compiler and thus are known as compile-time errors.

Most Frequent Syntax errors are:

1) Missing Parenthesis (})
2) Printing the value of variable without declaring it

* Logical Errors:-

⇒ On compilation and execution of a program, desired output is not obtained when certain input values are given

⇒ These types of errors which provide incorrect output

but appears to be error free are called logical errors.

⇒ These are one of the most common errors done by beginners of programming.

⇒ These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

* Run-time Errors :-

⇒ Errors which occur during program execution (run-time) after successful compilation are called run-time errors.

⇒ One of the most common run-time error is division by zero also known as division error.

* Linker Errors :-

⇒ These errors occur when after compilation we link the different object files with main's object using Ctrl+F9 key (Run).

⇒ These are errors generated when the executable of the program cannot be generated.

* Semantic Errors :-

This error occurs when the statements written in the program are not meaningful to the compiler.

```
// C Program to illustrate to semantic error
void main ()
{
int a, b, c;
a+b=c; // semantic error
```

error

error: lvalue required as left operand of assignment

Introduction to C Programming:-

* Structure of C Program:

Documentation Section

Link Section

Definition Section

Global Declaration Section.

Main() function section

(

Declaration Part
Executable Part

)

Sub program section.

Function 1
Function 2
.....
function n

* Documentation Section.

⇒ This section consists of a set of comment lines givin the name of the program, and other details which th Programmer would like to user later.

→ //---------- single line

→ /*---------

---------*/---------- mutliple lines.

Link Section

→ Link section provides instruction to the compiler to link functions from the system library.

Ex: #include <stdio.h>
#include <conio.h>

* Definition Section

→ Definition section defines all symbolic constants

Ex: # define A 10

* Global declaration section

⇒ Some of the variables that are used in more than one function throughout the program are called global variables and declared outside of all the function

⇒ This section declares all the user-defined functions

* Main ()function section

→ Every C program must have one main() function section. This contains two parts

* Declaration part
* Executable Part

* Declaration Part

→ This part declares all the variables used in the executable Part

Ex: int a,b;

* Executable Part

→ This part contains at least one statement.

⇒ These two parts must appear between the opening and closing braces

→ The program execution begins at the opening brace

→ at the closing brace

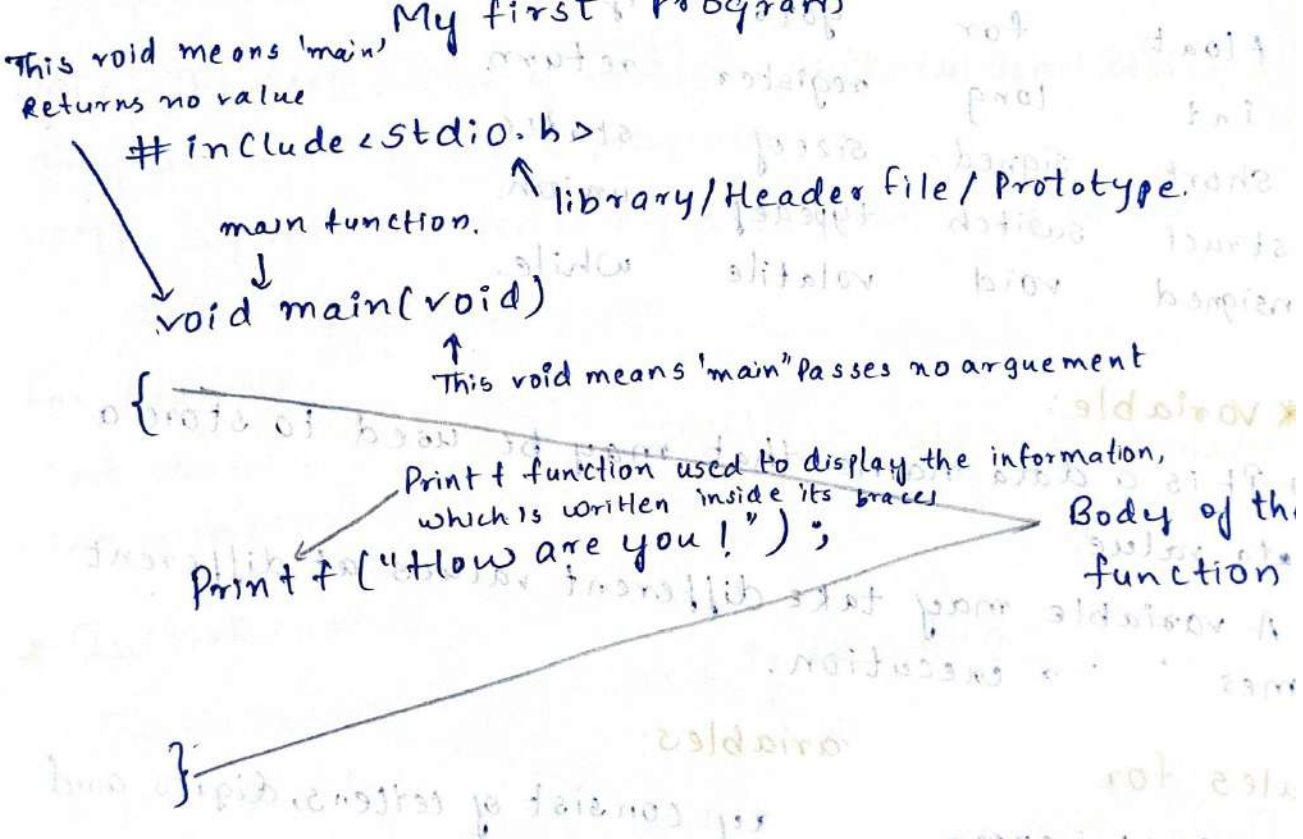
→ All the statements in the declaration and executable parts end with a semicolon (;)

* Sub program section.

⇒ This section contains all the user-defined functions, that are called in the main() function

⇒ User-defined functions are generally placed immediately after the main() function, although they may appear in any order

Example:



* C tokens

The C tokens are

1) keywords
2) variables
3) Identifiers
4) constants
5) datatypes
6) Operators

*Keywords :-

→ Keywords are also called as reserved one
→ There are 32 key words
→ All the keywords would be in lower case
→ A Keyword should not be a variable.

The following keywords

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

* Variable :

→ It is a data name that may be used to store a data value.
→ A variable may take different values at different times during execution.

Rules for naming variables:

→ variable names may consist of letters, digits and underscore (_) character
→ First char must be an alphabet or an '_'
→ Length of the variable cannot exceed upto 8 characters (few upto 31 chars)
→ White space is not allowed
→ variables name should not be a keyword
→ upper and lower case are significant.

Examples :-

Valid :
Mark, Sum

Invalid :
price $, qr

* Declaratio
→ Declarati
→ It tells th
→ It specifies

NOTE :- The d
they are u
→ The syn
data -

for exampl
int coun
int numb

* Initiali
→ Initiali
starting
→ C does
→ so if y
get unex
→ fortun
when yo

for ex
int n

Examples:-

Valid:
Mark, Sum1, tot_value, Delhi

Invalid:
Price $, group one, char

* Declaration of variables with examples:
→ Declaration does two things:
→ It tells the compiler what the variable name is
→ It specifies what type of data the variable will hold

NOTE:- The declaration of variables must be done before they are used in the C program.
→ The syntax for declaring a variable is as follows:

data-type $v_1, v_2, \ldots, v_n$;

for Example:
int count;
int number, total;

* Initialization
→ Initialize a variable in C is to assign it a starting value.
→ C does not initialize variables automatically.
→ So if you do not initialize them properly, you can get unexpected results.
→ fortunately, C makes it easy to initialize variab when you declare them

for eni
int n=45;

* Identifier :-

Identifier is the name of the variable (or) array (or) structure (or) Pointers (or) function

The rules for Identifier are same as variables

* Constants in C

→ C constants is the most fundamental and essential Part of the C programming language.

→ constants in C are the fixed values that are used in a Program.

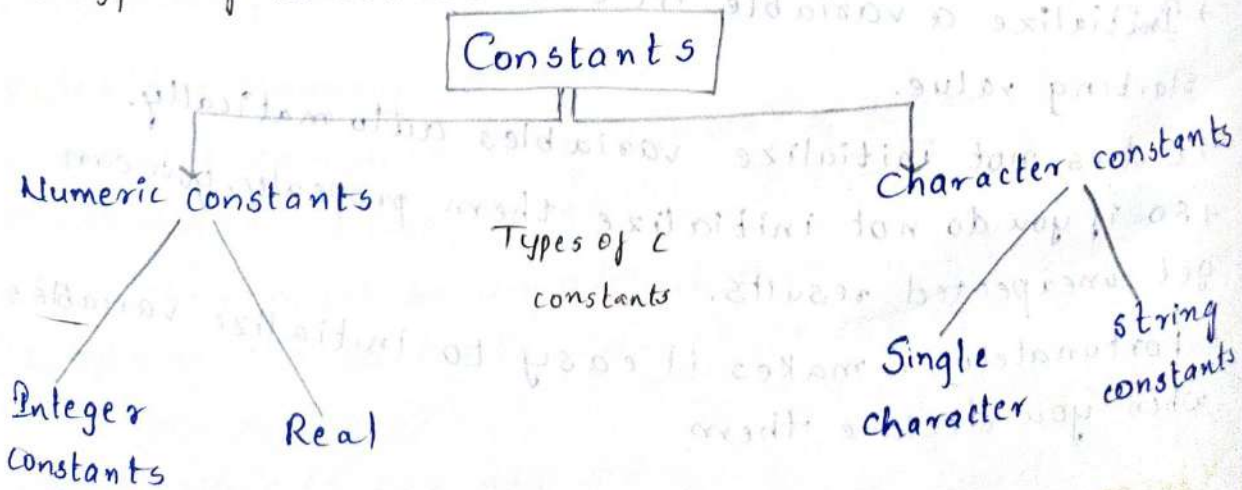
→ Its value remains the same during the entire execution of the program.

* Constants are also called literals.

* constants can be any of the data types

* It is considered best practice to define constants using only upper case names

* Types of constants:



* Types of c constants:

1. Integer constants
2. Real constants
3. Character constants
4. String constants

* Integer constants:

An integer constant refers to a sequence of digits
There are three types of integers, namely, decimal integer, octal integer and hexadecimal integer

Ex for integer constants:

426, +786, -34 (decimal integers)
037, 0345, 0661 (Octal integers)
0X2, 0x9F, 0x (hexa decimal integers)

* Real constants:

These quantities are represented by numbers containing fractional parts like 18.234
Such numbers are called real (or floating point) constant

Ex. for real constants:

+325.34
426.0
-32.67

* Single character constants

Single character constant contains a single character enclosed within a pair of single quote marks.

for ex: 'A', '5', '?', '+', '*'

* Backlash character constants:

C supports some special backslash character conta
that are used in output functions.
some of the back slash character constants are as
follows

Constant	Meaning (Name)
'\n'	Newline
'\r'	carriage return
'\f'	form feed
'\t'	horizontal tab
'\a'	alert
'\b'	backspace
'\0'	null
'\v'	vertical tab
'\\'	back slash
'\''	single quote
'\"'	Double quote

String constants:

String constant contains a group of characters enclosed within a pair of double quote marks

for ex:

"hello"

"XYZ"

"1234"

"@#$%"

* Datatype

→ A datatype

a variable

the type of

Program.C

may have

representa

→ C suppor

* Primar

* user-

* Derive

* Empt

fundam

(or) Prim

data t

Int, Dou

Float

Datat

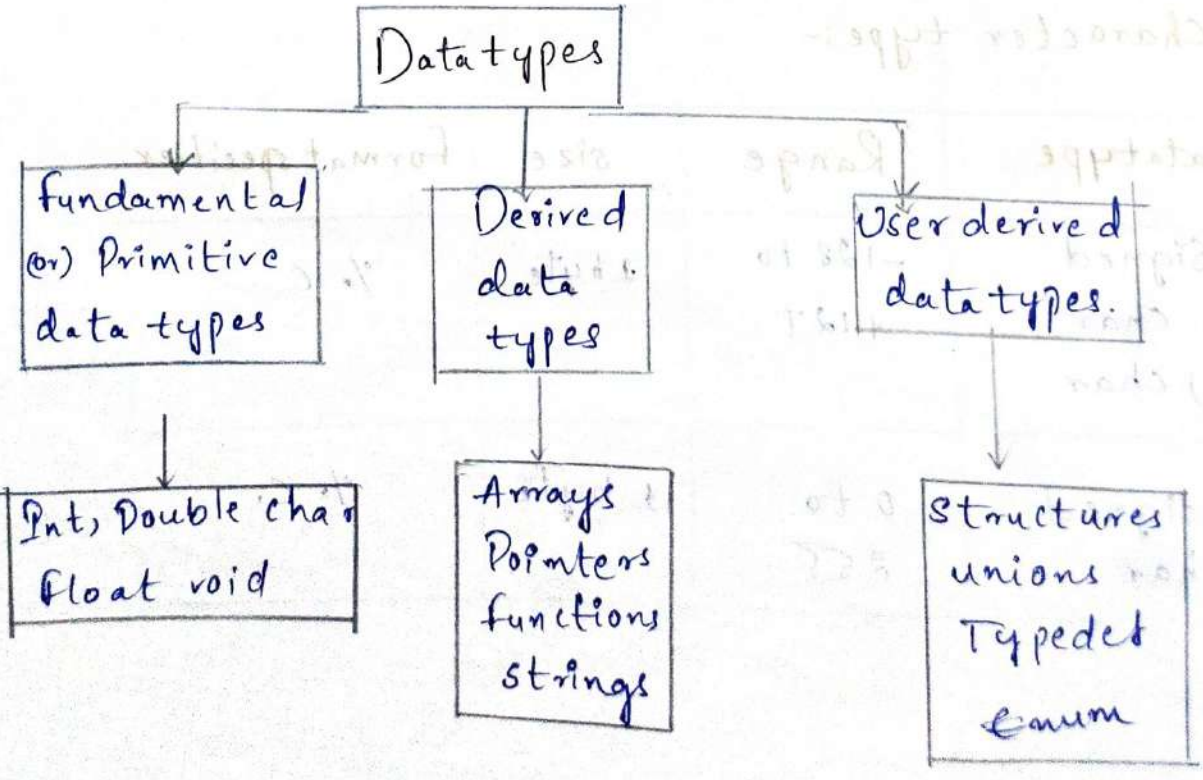
03-10-23

* Datatypes :-

→ A datatype is a classification of the type of data that a variable can hold in computer program. Datatype is the type of data that are going to access within the Program. C supports different datatypes. Each datatype may have pre-defined memory requirement and storage representation.

→ C supports 4 classes of data types :

* Primary (or) (fundamental) data type
  [int, char, float, double]
* user-defined datatype
  [typedef, structures, unions, enum]
* Derived datatype
  [arrays, pointers, strings, functions]
* Empty datatype [void]



fundamental (or) Primitive data types:

Datatype	Range	size	format specifier
1) Int	-2^{15} to $+2^{15}-1$ -32768 to +32768	4 bytes [on 32 bit Processors]	%d or %i
2) Signed short int (or) short int	-128 to +127	4 bytes	%d or %i
3) Unsigned short int	0 to 255	4 bytes	%d or %i
4) Unsigned int	0 to 65,535	4 bytes	%u
5) unsigned long int	0 to 4,294,967,295	4 bytes	%lu
6) long int or signed long int	-2147483648 to +2147483647	4 bytes	%ld

* Character type:-

Datatype	Range	size	format specifier
1. Signed char (or) char	-128 to +127	1 byte	%c
2. Unsigned char	0 to 255	1 byte	%c

* floating poin

Datatype = flo

size = 4 bytes

* Double prec

1. Data type = D

size = 8 by

2 Data type =

dou

size : 10 by

* floating point types :-

Data type = float
size = 4 bytes
Range = 3.4e-38 to 3.4e+38
format specifier = %f

* Double precision type :-

1. Data type = Double
size = 8 bytes
Range = 1.7e-308 to 1.7e+308
format specifier : %lf

2. Data type = long double
size : 10 bytes
Range : 3.4e-4932 to 1.1e+4932
format specifier : %Lf.

* C Operators:-

An operatorm is that which performs some op on operands.

There are 8 types of operators. They are

1) Arithmetic operators
2) Relational operators
3) Logical
4) Assignment
5) Increments and Decrement
6) Conditional
7) Bitwise
8) special.

* Arithmetic operators:-

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication.
/	Division
%	Modulus operator.

* Write the C program to implement arithmetic operators.

```
#include <stdio.h>
int main()
{
 int a,b
 scanf("%d%d", &a, &b);
 Printf ("Addition = %d\n", a+b);
 Printf ("Subtraction = %d\n", a-b);
 Print f ("Multiplication = %d\n", a*b).
 Printf("Division =
 Print f("Modulus =
 }
 return 0
```

* Relational Ope

Operator
<
<=
>
>=
==
!=

* Write the C p operators

```
#include <stdi
int main()
{
 int a, b;
 scanf("%d%d
 Printf("Great
 Printf("Great
 Printf("Less
 Printf("Less
 Printf("Equ
 Print f("Not
 }
```

* Logical O

Operator
&&
||
!

```
Printf("Division = %d\n", a/b);
Printf("Modulus = %d\n", a%b);
}
return 0
```

NOTE: Modulus operators can be operated on floating point values

* Relational Operators :-

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than orequal to
==	is equal to
!=	is not equal to.

* Write the C program to implement relational operators

```
#include <stdio.h>
int main()
{
  int a, b;
  scanf("%d%d", &a, &b);
  Printf("Greater than = %d\n", a>b);
  Printf("Greater than or equals to = %d\n", a a>=b);
  Printf("Less than = %d\n", a<b);
  Printf("Less than arequals to = %d\n", a<=b);
  Printf("Equals to = %d\n", a==b);
  printf("Not equals to = %d\n", a!=b);
}
```

* Logical Operators :-

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

* Write a C program to implement logical operators

```
#include<stdio.h>
int main()
{
  int a,b,c;
  Scanf("%d %d %d", &a, &b, &c);
  Printf("Logical AND = %d \n", ((a<=5)&&(c>b
  Printf("Logical OR = %d \n", ((a<=5) || (c>b+
  Printf("Logical NOT = %d, (!(a>b)));
}
```

output a=3, b=5, c=6

AND = 0

OR = 1, NOT = 1

Output

a = 3, b = 5, c = 6

AND = 0

OR = 1

NOT = 1

* Assignment Operators :-

'=' this is called assignment operator

Arithmetic operators when assigned combined with assignment operators will give you "short hand notations"

+	+ =	
-	- =	
*	* =	
/	/ =	
%	% =	

1) a = a+2 can be written as a+=2

2) a*=c-5 means a=a*(c-5)

*NOTE:- Differences between '=' & '=='

1) This is an assignme
operators

2) a=10 means
constant 10 is
to variable

* write a C
operators.

#include <s
int main()
{
int a;
scanf(
Printf(
Printf
Printf
Printf
Print
Print
}

Output:

10
assignmen
15
30
10
4
0

* Increm

⇒ Increm

⇒ Decre

`=`	`==`
1) This is an assignment operators	1) This is a relational Operators.
2) a=10 means constant 10 is assigned to variable a	2) a==10 means we are checking whether the value of a is 10 or not

* write a C program to implement assignment operators.

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d",&a);
    Printf("Assignment a= %d\n", a);
    Printf("%d\n", a+=5);
    Printf("%d\n", a*= 2);
    Printf("%d\n", a/= 3);
    Printf("%d\n", a-= 6);
    Printf("%d\n", a%= 4);
}
```

Output:

```
10
assignment a=10
15
30
10
4
0
```

* Increment / Decrement operators:

⇒ Increment operators will increament the value b

⇒ Decrement Operators will decrement the value by

* There are 2 types of increment and decrement they are Pre and Post

	Incre	decr.
Pre	++a	--a
Post	a++	a--

	a	b
b = a++	11	10
b = ++a	11	11
b = a--	9	10
b = --a	9	9

*Write a C program increment/decrement operator

```
#include<stdio.h>
int main()
{
int x,a,b,c,d;
scanf("%d",&x);
a=x++;
b=++x;
c=x--;
d=--x;
Printf("Post increment of x=%d\n",a);
Print f("Pre increment of x=%d\n",b);
Print f("Post decrement of x=%d\n",c);
Print f("Pre decrement of x=%d\n",d);
}
```

Output: a=10.
10 → Post increment
12 → Pre increment
12 → Post decrement
10 → Pre decrement

* Bitwise oper

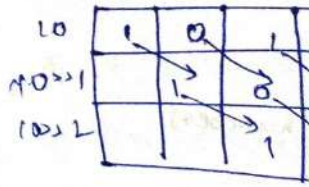
Operator
&
|
^
<<
>>

Ex:- &&

① 7 & 9
7 - 0 1 1 1
9 - 1 0 0 1
0 0 0 1 -

③ 7 ^ 9
7 - 0 1 1 1
9 - 1 0 0 1
1 1 1 0 -

⑤ 10 >> 2



⇒ When le
When ri

* Write a
- operato

* Bitwise operators :-

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive
<<	shift left
>>	shift right

Ex:- && ||

① 7 & 9

7 - 0 1 1 1
9 - 1 0 0 1
0 0 0 1 - 1

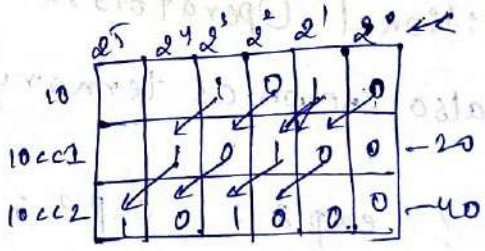
② 7 | 9

7 - 0 1 1 1
9 - 1 0 0 1
1 1 1 1 - 15

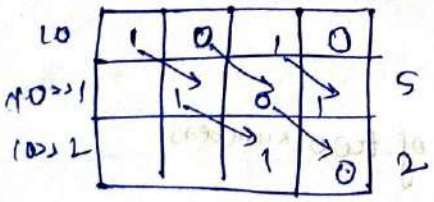
③ 7 ^ 9

7 - 0 1 1 1
9 - 1 0 0 1
1 1 1 0 - 14

④ 10 << 2.



⑤ 10 >> 2



→ When left shift is performed, the value gets doub
When right shift is performed, the value will be halv

* Write a C program to implement bitwise operators.

```
#include<stdio.h>
int main
 {
  int a,b;
  scanf ("%d %d", &a, &b);
  Printf("%d\n", a & b);
  Printf("%d\n", a | b);
  Printf("%d\n", a ^ b);
  Printf("%d\n", a<<2);
  Printf("%d\n", b>>2);
 }
```

Output:-

0
15
15
40
1

* Conditional Operators:-

* It is also known as ternary operator.

exp 1 ? exp 2 : exp 3 ;

where, expression 1 is the condition to
exp 2 will be evaluated if exp 1 is true.
exp 3 will be evaluated if exp 1 is false

* write a C program to find the largest of two numbers using ternary operator

```
#include<stdio.h>
int main()
{
  int a,b,c;
  scanf ("%d %d", &a, &b);
  //c = a>b ? a:b;
  // Printf("%d is largest", c);
a>b ? Printf("%d os largest",a) : Printf("%d is largest", b);
```

Output:
6 10
10 is larges

* Special
* comma o
* size of
* Pointer
* member
* Comma
value = (x
In for loo
for (n=
In while
while
Exchangi
t = x
* size of
m = size
n = size
K = size
* Expr
⇒ An ex
variables
Ex.
⇒ Most
conten
⇒ A sta
with a
Ex.

Output:

6 10

10 is largest.

* Special operators:-

* comma operator
* size of operator
* Pointer operators (& and *)
* member selection operators (. and ->)

* Comma Operator:-

```
value = (x=10, y=5, x+y);
```

In for loops:

```
for (n=1, m=10, n<=m; n++, m++)
```

In while loops

```
while (c=getchar(), c!='1')
```

Exchanging values

```
t=x, x=y, y=t;
```

* size of operator

```
m=sizeof(sum);
n=sizeof(long int);
K=sizeof(235L);
```

* Expressions:-

⇒ An expression in C is some combination of constants, variables, operators and function calls.

Ex:
```
c=a+b
tan(angle)
```

⇒ Most expressions have a value based on their contents

⇒ A statement in C is just an expression terminated with a semicolon.

Ex:
```
Sum=x+y+z;
```

* Precedence :-

⇒ Precedence rules decide the order in which different operators are applied.

⇒ Associativity rule decides the order in which multiple occurrences of the same level operators are applied.

* Hierarchy of operators in C.

⇒ The higher the position of an operator is, higher is its Priority. when an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.

* Associativity can be of two types - left to right or right to left.

⇒ Left to right means, as you go from left to right in an expression which operator among the two is found, execute it first. Right to left means, as you go from right to left in an expression which operator among the two is found, execute it first.

* consider expression a = 3/2*5 Here there is a tie between operators of same priority, that is between / and *

* This tie is settled using the associativity of / and *. But both enjoy left to right associativity, which means as you go from left to right / is found so execute it first.

* The -fo
Precedenc

Rank
1
2.
3.
4.
5.

* The following tables shows us associativity and Precedence of operators in C.

Rank	Operators	Meaning	Associativity
1	++	Postfix increment	Left to Right
	--	Postfix decrement	
	()	function call.	
	[]	array subscripting	
	.	structure and union member access.	
	→	structure and union member access through pointer	
2.	++	Prefix increment	Right to left
	--	Prefix decrement	
	+	Unary plus.	
	-	Unary minus.	
	!	Logical NOT	
	~	BITWISE NOT	
	(TYPE)	TYPE CAST	
	*	Indirection	
	&	adress of	
	size of()	size of	
3.	*	MULTIPLICATION	LEFT TO RIGHT
	/	DIVISION.	
	%	MODULO DIVISION	
4.	+	ADDITION	LEFT TO RIGHT
	-	Subtraction	
5.	<<	Bitwise shift left	LEFT TO RIGHT
	>>	Bitwise shift right	
6.	<	Relational operators lesser	LEFT TO RIGHT
	<=	lesser than or equal to.	

	>	Greater than	
	>=	Greater than or equal to.	
7.	==	is equal to.	LEFT TO RIGHT
	!=	is not equal to.	
8.	&	BITWISE AND	LEFT TO RIGHT
9.	^	BITWISE XOR	LEFT TO RIGHT
10.	|	BITWISE OR	LEFT TO RIGHT
11.	&&	Logical AND	LEFT TO RIGHT
12.	||	Logical OR	LEFT TO RIGHT
13.	?:	Ternary	left to right
14.	=	Assignment operator simple	RIGHT TO LEFT
	+=	Assignment by sum	
	-=	Assignment by difference	
	*=	Assignment by product	
	/=	Assignment for quotient	
	%=	Assignment for remainder	
	<<=	Assignment by bitwise left shift	
	>>=	Assignment by bitwise right shift	
	&=	Assignment by bitwise AND.	
	^=	Assignment by bitwise XOR	
	!=	Assignment by bitwise OR	
15.	,	COMMA	LEFT TO RIGHT

* Type Co

⇒ The process
type to an

* There are 2

1) Automati

2) Casting

⇒ The rules

1. Integer t
2. Signed ty
3. Short nu
4. double > f

* Implicit

* write a C
conversion

```
#include<s
void main(
{
int a=5,b
float r;
r=a/b;
Printf("r
}
```

07-10-23

* Type Conversion :-

⇒ The process of converting the variable from one data type to another is known as type conversion.

* There are 2 types of conversion

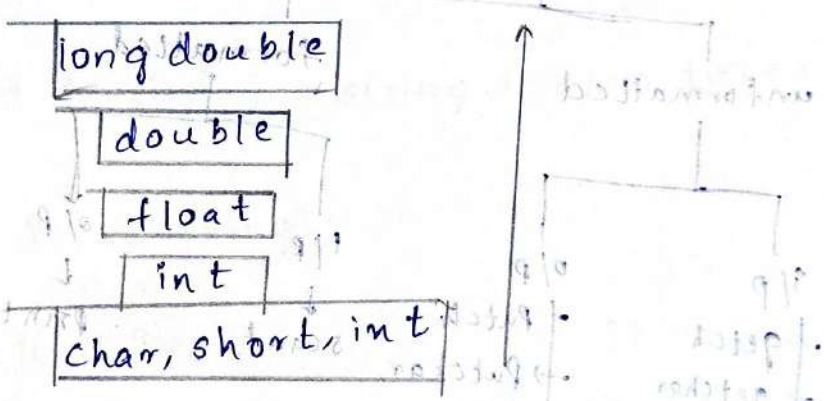
1) Automatic or implicit

2) Casting or explicit.

⇒ The rules that have to be followed.

1. Integer types are lower than floating point types.
2. Signed types are lower than unsigned types.
3. Short number types are lower than longer types.
4. double > float > long > int > short > char.

* Implicit conversion.



* write a C program to implement implicit type conversions.

```
#include<stdio.h>
void main()
{
int a=5,b=2;
float r;
r=a/b;
Printf("r=%.f\n",r);
}
```

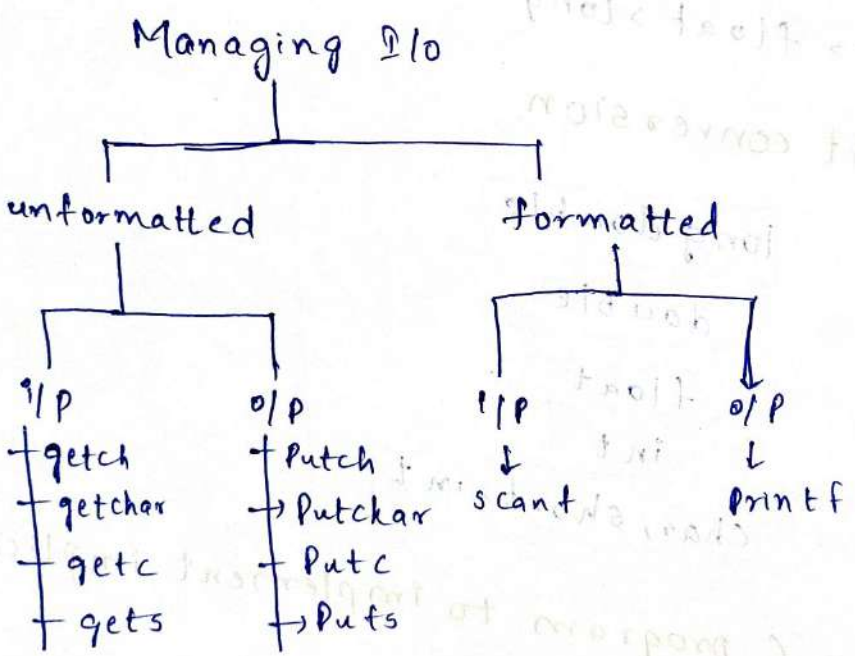
output

r= 2.000000.

* Write a C program to implement explicit type conversion.

```
#include<stdio.h>
void main()
{
int a=5, b=2;
float r;
r=(float)a/b;
Print("r=%.f\n", r);
}
```

* Managing I/O :-



* Unformatted I/O :-

*getchar() :-

It is used to accept a single character using library function in a c program

Syntax :- variable-name = getchar();

*Putchar() :-

Single characters can be displayed using the library function putchar()

Syntax: Putchar(variable-name);

*gets() :

This function is used to read a string from the keyboard if input device is not specified

Syntax: gets(string);

*Puts() :-

It is used to display a string, it also takes single argument

Syntax :- Puts(string);

* formatted I/O :-

⇒ The scanf() function is used to input data in a formatted manner.

Syntax :- scanf("control string", list of adress of variables);

(or)

scanf("control string", &var1, &var2, ... , &varn);

⇒ In C, to represent an address of any location, an ampersand (&) is used

⇒ scanf() ignores all leading spaces, blanks, tabs, newlines etc

⇒ scanf() can also used to read a string of characters

⇒ C provides inbuilt function in library stdio.h known as printf()

⇒ The printf() moves data from computers memory to standard output device.

Syntax: Printf("control string", var1, var2, ---, varn);

The control string entries are usually seperated by spaces and precedence.

* Write a C program to print integers in different formats.

```
#include<stdio.h>
void main()
{
 int a=1234;
 Printf("%d %d\n", a,a);
 Printf("%.2d\n", a);
 Printf("%.9d\n", a);
Printf("%.09d\n", a);
Printf("%-9d%d\n", a,a);
 }
```

Output:

```
1234 1234
1234
     1234
000001234
1234      1234
```

* Write a C progrom to implement floating point values.

```
#include <stdio.h>
void main()
{
 float a=1234.56;
 Printf("%f\n", a);
 Printf("%2.2f\n", a);
 Printf("%2.1f\n", a);
 Printf("%9.2f\n", a);
 Printf("%9.1f\n", a);
 Printf("%09.2f\n", a);
 Printf("%-9f\n", a);
 }
```

Output:-

```
1234.560059
1234.56
1234.6
  1234.56
   1234.6
00001234.56
1234.560059.
```