5/9/22

PLACEMENTS:

Programming Lang.:
Used to instruct the machine to perform some specific tasks.

Levels:

1)- Binary (0's & 1's) Lang.
⇒ Machine understands only this language (0's & 1's)

0 ⇒ Low
1 ⇒ High

→ Machine can und. : (ASCII) with help of
∴ (A-Z) ∴ (a-z) ∴ (0-9) ∴ spl. char. (-/,)
→ ASCII ⇒ American Std. code for Information Interchange
(7bit)
∴ It is an encoding technique, which is used on all levels.

A-65 a-97 0-48

Z-90 z-122 9-57

→ Humans can't und. or can develop logic using only 0's & 1's.

→ So, Assemblers was invented to convert Assembly level language into 0's & 1's. (pre-defined)

→ Assembly level language consists basic English commands, which was und. by IC's (Integrated Circuits) and Humans. / Microprocessors
→ such as ADD, SUB, MVL, MOV, etc.

→ Disadv.:
.. STILL Lang. was abstract.
.. Looping was a prob. in Assembly lang.

→ Then, to overcome Disadv. of Assemblers, Compilers were introduced, to convert High-level lang. to 0's & 1's.

→ High-level-lang. are easy to learn & overcomes Disadv. of Assembly lang. eg: C/C++, Java, Python

→ (Basically a S/W)

→ Diff. High-level-languages has it's own compilers.
eg: C - GCC comp.
Java - JVM comp.

Questions:
(i) What is C?
(ii) C is a ___ lang.

→ C is a Upgradation of Assembly-level languages.

→ C is a Mid-level or Assembly level lang. But, it is considered as

Q's:

- Why C is a Mid-level lang?
- Diff. Assembler, Compiler, Interpreter.

→ C was developed ~~as~~ by Dennis Ritchie in Basic labs.
→ C was developed in 1972.

(Q) Features of C: (HW)

→ General purpose language.
→ Portable - Dependent
→ Built-in Header files & Func.'s
→ Mid-level / Assembly - level
→ Statically Typed
→ Faster Execution.
→ Closer to H/w & has more control over H/w

Structure of C:

① Header files.
② Global var.
③ Functions
④ int main()
{
⑤ Declaration
⑥ statements.
return 0; // opt Depends on Data type
}

#
↳ preprocessor-directives
Libraries are present in a diff. loc. so, we include header files.

↳ It is performed before execution of a Program.

Page No.
Date / /20

eg program:

prog1.c

```
#include <stdio.h>

void main ()
{ printf ("Hello World");
}
```

→ IDE ⇒ Integrated Development Environment
eg: VS code, Dev-C++, etc.

Execution-Flow:

(1) → Compiler compiles the code into an byte-code file.

(2) → This byte-code file is converted by Interpreter into 0's & 1's.

(3) → After Job is done, output file is produced.

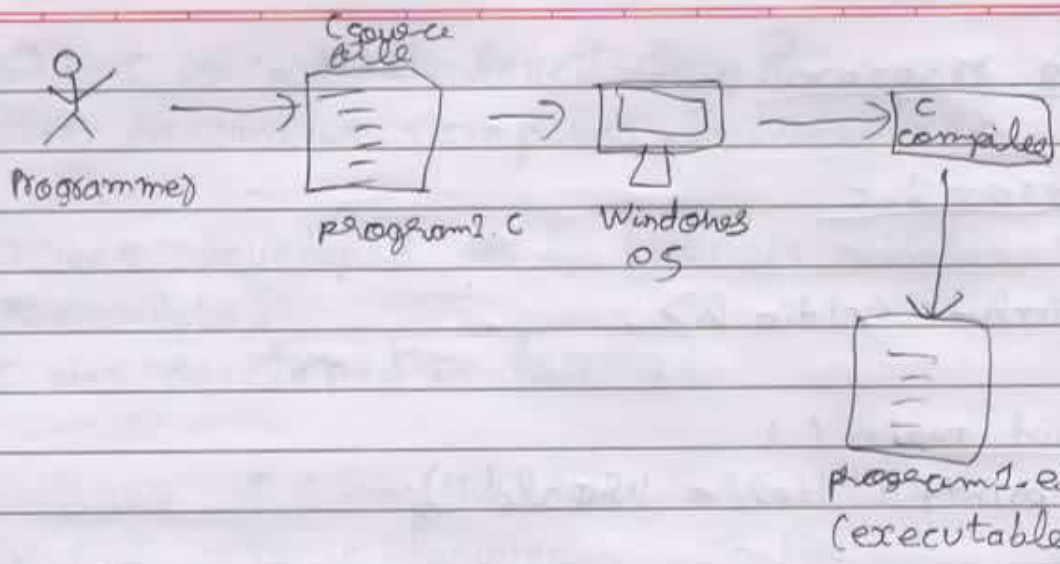
(1) ⟶ (2) ⟶ (3)

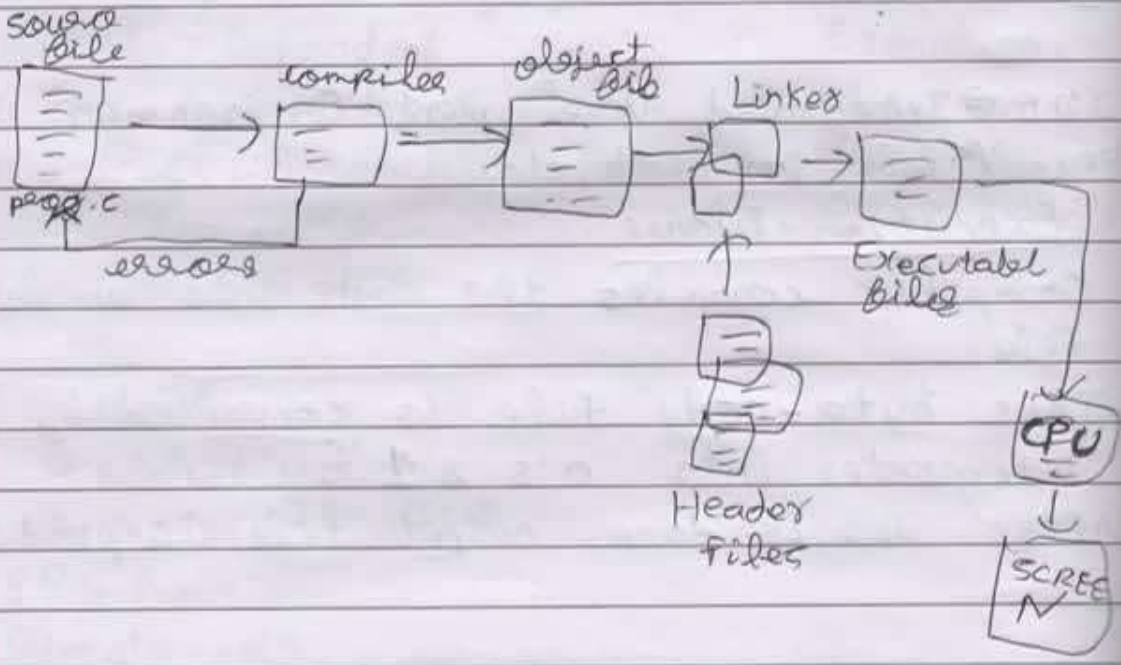
source file	obj file	output file

~~Execution Flow of C programs~~

→ C is platform-dependent,

File generated on 1 platform, will be executed on same platform(OS). This is platform-dependency.





Token:

→ It is the most basic/smallest unit of a program.

OR

→ It is a Building Block of Program.

→ Types of Tokens:

(i). Keywords (Pre-def. words)

(•. We can't change keywords, they are fixed.

•. Each Keyword has it's own task.

•. lowercase letters.

eg: void, int, break, continue, return.

∴ There are 32 Keywords:

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

(ii) Identifiers:

→ It is the naming conventions for variable, function, arrays, structures, etc.

(iii) Constants:

→ It is a value, which can't be changed.

Rules for identifiers:

1). can't start with digits.
2). Keywords can't be used
3). Spl. char except (—) can't be used.
4). White spaces not allowed. (_, $).

(iv) Strings:

→ It is a seq. of characters.
→ It is a literal.
→ It is enclosed within " ".

→ In C, string is represented as array of charecters, which ends with null charecter (\0).

(v) Special charecters :

; ⇒ End of stmt.
: ⇒ label
() ⇒ func., Expressions
{} ⇒ Block of stmt.
⇒ pre-processor directive
[] ⇒ Array
, ⇒ Seperator
* ⇒ Asterisk, ⇒ Pointer, multiply.

(vi) char:

→ It is literal
→ It is enclosed within ''.
→ Always, length ⇒ 1.

printf() & scanf() ⇒ Present in <stdio.h>

printf() ⇒ prints o/p
scanf() ⇒ takes i/p from user

printf("%d", list of var);
scanf("%d", list of var);
↑ format specifier

Page No.
Date / /20

Format Specifiers:

%c	single character
%s	string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double.
%n	prints nothing
%d	decimal integer (base ⇒ 10)
%i	decimal int + base
%o	octal (base 8) integer
%x	Hexadecimal (base 16) integer
%p	an address (or pointer)
%f	Float
%u	unsigned int
%e	floating point no. in scientific notations
%E	—
%%	% symbol

\ ⇒ (Back slash) Escape Sequence
/ ⇒ Sp. character

(1) Escape Sequence:

\n	newline
\t	tab
\b	backspace
\r	carriage return
\a	Audible bell
\'	printing single quote
\"	printing double quotes

\? question mark
\\ ~~Black~~ Back slash
\f Form seed
\v Vertical Tab
\0 null characters
\nnn octal
\xhh Hexadecimal value

\8 mechanism:

\8 <-> sp

Good Evening ~~to~~ bigsale
7
-
7 here

o/p:
bigsalening

Comments:

→ They are statements, that are ignored by compiler.
→ It is written for Programmer's Reference.

single-line: //
multi-line: /* */

Line-Splicing:

```
//printf("Hi"); #\
printf("st.1");
printf("st.2");
```

o/p:
st.2

At end of single line comment, if we use a \, it merges the immediate next line.

6/9/22

Datatypes:

It is used to assign/specify the type of data of a variable.

Types:

(i). Basic:

int, float, char, double

(ii). Derived:

Array, pointer, structure, union

(iii). Enumeration (Enum):

enum

(iv). void

Memory Size & Range of Data Types:

Program to add 2 nos:

```
#include<stdio.h>

void main()
{ int n1,n2,sum;

 printf("Enter two nos : \n");
 scanf("%d %d",&n1,&n2);
 sum = n1+n2;
 printf("sum = %d \n", sum);

}
```

Memory Range for 16 bit:

		Range	Bytes
char	signed char	-128 to +127	1
	unsigned char	0 to 255	1
int	short signed int	-32768 to 32767	2
	short unsigned int	0 to 65535	2
	signed int	-32768 to 32767	2
	unsigned int	0 to 65535	2
	long signed int	-2147483648 to 2147483647	4
	long unsigned int	0 to 4294967295	4
	float	-3.4e38 to +3.4e38	4
	double	-1.7e308 to +1.7e308	8
	long double	-1.7e4932 to +1.7e4932	10

Page No.
Date / /20

Variables:

It is the name given to a Memory Location.

Syntax of Declaration:

```
datatype variable_name;
```

Syntax of Definition:

```
datatype
variable_name = value/variable/exp;
```

Scope of Variables:

Visibility of variables is known as scope of variables

Types of variables:

1). local variables:
They are present in a block or main().
They can be accessed only in that scope.

2). Global variables:
They are declared outside main().
They can be accessed anywhere in that program.

```
#include <stdio.h>
int b=90; // GV

int main()
{ int a=50; // LV
  printf("%d",a);
  printf("%d",b);
}
```

o/p:
5090

P1.c:

```
#include <stdio.h>
int a=90;
Void main()
{  printf ("%d",a);  //90
   int  a=10;   //wrong decl.
   printf ("%d",a);  //10
}
```

O/P:
9010

```
#include <stdio.h>
int a=90;
void main()
{   {
      a=99;
    }
   printf ("%d\n", a);
}
```

O/P:
99

```
#include <stdio.h>
int a=90;
void main()
{ int a=100;
   {
     a=99;
   }
  printf ("%d\n", a=200);
}
```

O/P
200

```
main()
{
  printf("%d", int a=10); //wrong dec.
}
```

*** Only assigning is allowed.
In C, only declaration is allowed at top in main().

Static Variables:
static keyword is used for declaring static variables.

External variables:
- extern keyword is used for External variables.
- These variables can be shared within multiple C files.

Automatic variables:
- auto keyword is used Automatic variables.
- All local var. are automatic variables, by default.

Operators:

It is a pre-defined symbol, used to perform a specific operation.

sum = a + b → operands
↓
operator

Operator Types:

(i) Unary operator: 1 operand
(ii) Binary operators: 2 operands
(iii) Ternary operators: 3 operands

Unary Operator:

++, --
Inc, Dec

	Inc.	Dec.	
pre	++a	--a	pre
post	a++	a--	post

Binary Operator:

+, -, *, /, %	Arithmetic
<, <=, >, >=, ==, !=	Relational
&&, ||, ! (Tech. not possible)	Logical
&, |, <<, >>, ~, ^	Bitwise
=, +=, -=, /=, *=, %=	assignment

Ternary Operator:

?: Ternary

eg:

```
c = (a>b)? a : b
```

Equivalent Logic

```
if(a>b)
    return a;
else
    return b;
```

Pre: Update it and Use it.

Post: Use it and Update it.

Pre: Up ⇒ U

Post: U ⇒ Up

```
# include <stdio.h>
void main()
{ int a;
  printf("%d", a);
}
```

O/P:
0

*** In C, By default 0 is assigned
For char data type, nothing is printed.
{ (space or ? is printed)

Pre V/s Post:

```
#include <stdio.h>
void main()
{ int a=10; //
  a = a++;  // a = 10
  printf("%d", a); //10
}
```

Assignment over takes updation here

- Here value of a=10 didn't update to 11, ~~as we didn't use 'a' in any expression.~~ as assignment of a = a++ over takes updation part of a++,

```
#include <stdio.h>
void main()
{ int a=10;
  a++; // 11
  printf("%d", a); //11
}
```

```
main()
{ int a = 10;
  int b = a++ + a;    // 10 + 11
  printf ("%d \n %d", a, b); // 11, 21
}
```

O/P:

```
11
21
```

```
main()
{ int a = 10;
  int b = a++ + a + a++; // 10 + 11 + 11
  printf("%d \n %d", a, b); 12, 32
}
```

O/P:

```
12
32
```

Rules: (post-fix)
1. Use (U)
2. Update (Up.)

Rules: (pre-fix)
1. Update (Up.)
2. Use (U)

```
#include <stdio.h>
void main()
{ int a = 10;
  a = ++a; // 11
  printf("%d", a); // 11
}
```

Page No.
Date / /20

```
#include <stdio.h>
Void main()
{ int a=++3; //syntax error
  printf("%d", a);
}
```

```
#include <stdio.h>
Void main ()
{ printf("%d", 4++); //syntax error
}
```

Post/Pre increment/decrement op. can only used on variables.

Q: Interview Q:

(Ans)

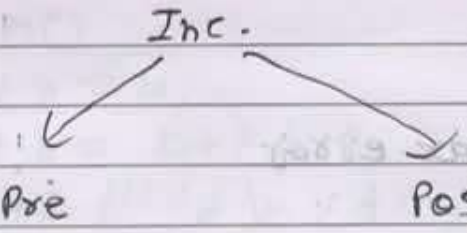
```
main()
{ int a=20;
  a = ++a + ++a + a; // 21+22+22 65
  printf("%d", a);    +1 //66
}
```

~~main()~~

```
{ int b=20;
  b = a++ + a++ + a; // ~~20+24+25~~ 67+68+68
  printf("%d", b);    //+1 201
}
```

```
main()
{ int a=20;
  a=a++ + a++; //20+21
  printf("%d", a); //41
}
```

```
main()
{ int a=20;
  a=a++ + ++a; //20+22
  printf("%d", a); //42
}
```




```
main()
{ a=20;
  a=a--; // 20
  printf("%d", a); //20
}
```

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

O/P:
0

Relational operator:

It is used to perform comparision b/w new value and old value.

Page No.
Date / /20

Logical Operators:

It returns either 0 or 1.

0 ⇒ False

1 ⇒ True

Types:

NOT (!)

AND (&&)

OR (||)

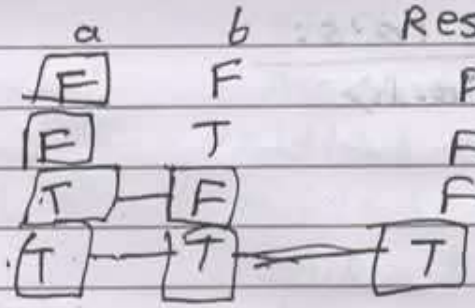
AND (&&):

It returns 1, when both i/p's are True.

OR (||):

It ~~returns~~

Truth Table:



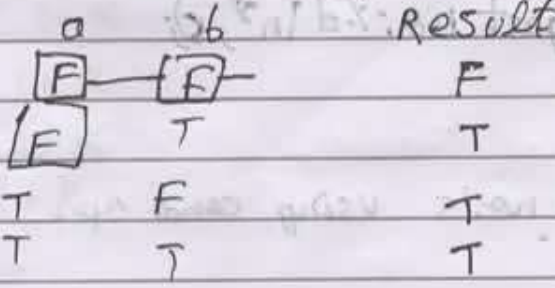
Syntax:

$OP_1 \&\& OP_2$

OR (||):

It returns 1, when ~~to~~ any one i/p is True.

Truth Table:



Sy.:

$OP_1 \,||\, OP_2$

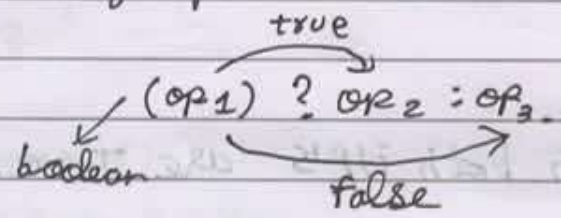
(NOT)

! op1:

→ It is unary op.
→ It reverses the o/p.

Conditional operator:

→ Ternary op.



(op1) is generally a condition or an expression.

Largest of 2 no.'s:

```
#include <stdio.h>
void main()
{ int a,b,c;

 printf("Enter 2 numbers: \n");
 scanf ("%d %d", &a, &b);

  c = (a>b) ? a:b;

  Printf ("Largest no: %d \n", c);
}
```

Q. Largest of 3 no.'s using cond. op.

Page No.
Date / /20

Assignment operators:

→ It is used to assign a value to a variable, which is declared.

% → Right-to-Left operator

Syntax:

```
Variable = Value;
```

7/9/22	* / %	left to right	Multiplication, Division, Modulus
	+ -	L to R	Addition, Sub
	<< >>	L to R	Bitwise Left shift & Right Shift
	=	R to L	Assignment
	+= -=		Add/sub assign.
	*= /=		Mul/sub assign.
	%= &=		Modulo & Bitwise assign.
	^= |=		Bitwise exclusive/inclusive OR assign.
	<<= >>=		
	? :	R to L	Ternary operator
	&	L to R	
	^	L to R	
	|	L to R	
	&&	L to R	

precedence

++ --		++ (Post)
! ~		++ (Pre)
(type)	R to L	
*		
&		
size of		

```
main()
{ int a=5;
printf("%d %d %d", ++a, a, a++);
}
```

~~O/P~~
~~6 6 5~~

First, Post takes priority (a=5)
Then, pre takes priority (a=7)

Flow:

a++ = 5
+1 (6)
++a = 7
a = 7

O/P

7 7 5

→ In, Precedence Table, Post > Pre.

Page No.
Date / 120

```
main()
{  int a = 5;
   int b = ++a + ++a;
   printf ("%d %d", a, b);
}
```

O/P:
7 14

```
main()
{ int a=5;
  int b = ++a + ++a + ++a;
  printf ("%d %d", a, b);
}
```

O/P:
8 22

Pre:
1). Update (Up.)
2). Use (U)
3). Assign (ass.) (Last Step) (R to L)

$b = \underset{(6)}{++a} + \underset{7}{++a} + \underset{8}{++a}$;

only 1st operand is dependent on the var. b.

*** Not all value of a is updated.

```
main()
{ int a=9;
  int b = ++a + ++a + ++a + ++a;
  printf ("%d %d", a, b);
}
```

10 + 11 + 12 + 13 = 47

O/P:
47 947

Compound Assignment operator / Shorthand operator:

→ Shorthand operators are abit fastes than normal exp.

eg: Balance = Balance - 25000;
Balance = 5000;

⇓

Balance -= 5000;

```
main()
{ int a=90;
  a /=10;   // a = a/10 = 9
  printf("%d", a);
}
```

O/P:
9

```
main()
{ int a=90;
  a += (a+a); // a = a + (a+a)
  printf("%d", a);
}
```

a = a + (a+a)
= 90 + (180)
a = 270

O/P
270

Page No.
Date / /20

Decision Statements:

1. if stmt.:

```
main()
{ printf("main begins");
  int a=10;
  if (a==10)
  { printf("if block begins \n");
    printf("equal");
  }
  printf("Main Ends");
}
```

2. if-else:

```
main()
{ int a,b;
    printf("Enter two nos: \n");
    scanf("%d %d", &a, &b);

    if (a>=b)
        printf("Largest Number: %d \n", a);
    else
        printf("Largest Number: %d \n", b);

}
```

```
if (cond)
  { stmt.
else
  { stmt.
  }
```

⇒ WET ⇒ Write, Execute, Trace

else if ladder:

```
if (condition 1)
{ // stmt. 1;
}
else if (condition 2)
{ // stmt 2;
}
else if (condition 3)
{ // stmt 3;
}
else
{ //stmt 4;   // optional
}
```

If all cond- is false in else if ladder, all blocks are skipped.

```
if ( (year %4 == 0 && year % 100 != 0) || year
                                    %400 == 0
    Leap year

else
    Non Leap year
```

Page No.
Date / /20

Switch:

```
switch (value/Expression/var)
{ case value/exp : {stmts}
    break; // recommended & Not Mandatory.

  default : {stmt.}
}
```

Interview: Q:

Note: • In switch, we can pass value of int, char ~~type~~ only. (ch var.)

•• Escape sequences shouldn't be used in scanf().

•• Variables can't be used for case.

Loops: •• Repeating a set of stmts., until cond. is satisfied.

•• for
•• while •• Skips some blocks, if cond. is not satisfied.
•• do while

•• while:

```
while (cond.)
{

}
```

if (1) ⇒ True
if (0) ⇒ False
if () ⇒ error

```
void main()
{ int i=1;                    // Initialization
  while (i<=10)               // Condition
  { printf ("%d \n", i);
    i++;                      // Updation
  }
}
```

*** Inside loops, use only variables.

The process repeats until cond. becomes false

O/P:

2: Q- WAP Cprog. to print even no.'s 2 to 10.

2. odd no.'s b/w 1 to 10.

- do while:

```
// Initialisation
do {
    stmts;
    // updation
} while (condition);
```

Q. WAP to print even using do while

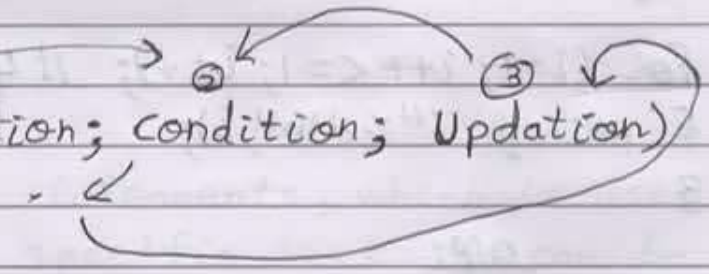
Page No.
Date / /20

** Tell what Interviewer wants, Don't Tell Extra.

Note:
In do-while loop, updation happens before while( ) part.

Q. Differentiate b/w while and ~~do~~ Do-while loop.

for loop:




```
for(initialization; condition; Updation)
{ statements.....
}
```

→ In for loop, we have 3 segments: ① ② ③

Int. Q:

```
main()
{ for(;;) //Infinite Loop
  { print("Hi");
  }
}
```

→ Segments are not Mandatory, they are recommended.

→ If condition or updation segment is not present, then, it considered as an infinite for loop.

→ In C=99 mode, for (int i =) isn't allowed.

```
for(i=1; ; i++); //allowed

for (i=1; i++ <=1; i++) // ? 2
{ printf ("%d\n", i);
}
```

O/P: 2

```
for (i=1; i++<=1; i++); // 4
{ printf ("%d\n", i);
}
```

O/P: 4

*** Sp. case:

```
for (i=0; i<5; i++);
    { printf ("%d ", i); }
```

Here, first for loop is processed, then exec. flow comes to printf stmt.

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

O/P		cond.	O/P:
2	1	2<=10	2
3	for	4<=10	4
4		6<=10	6
5		8<=10	8
6		10<=10	10
7			
8			

```
for (i=1; i++<=10; i++)
    print("%d\n", i)
```

O/P:	Comparisions:
2	1<=10
4	3<=10
6	5<=10
8	7<=10
10	9<=10

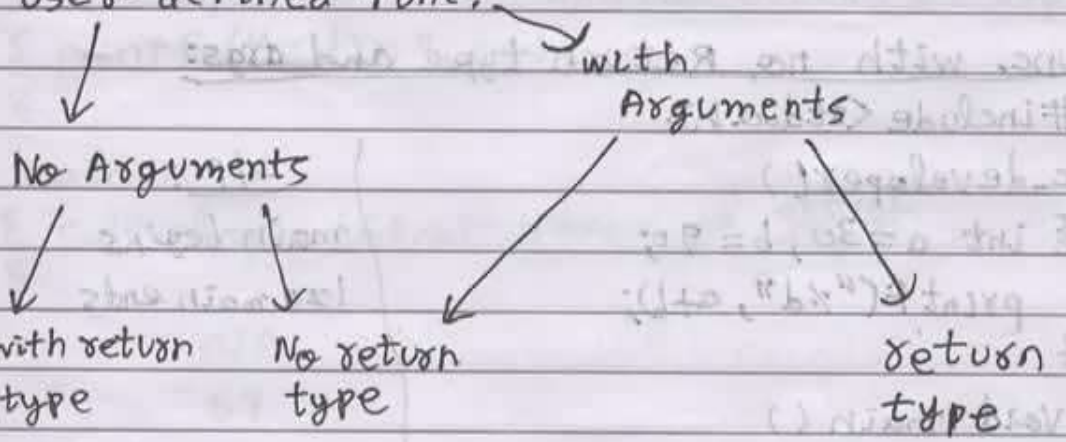
8/9/22 **Function:**

→ It is a set of statements, which is used for performing a specific task. It can be called anywhere in a program.

2 Types:

(i). In-Built Functions / Lib. Functions

(ii). User-defined func.



Func. Declaration: It is declared globally in C language.

```
Return-type func. name (Arg. name);
{
    statements; // Body
}
```

Func. Def. : Stmt.s inside func.

```
Return-type func. name (Arg. name)
{
   Body; // stmt.
}
```

Func. Call: used for calling a func., anywhere in program (main() inside).

```
func_name (Arg.);
```

→ If datatype is not mentioned, compiler assumes it to be void datatype.

Func. with no Return type and args:

```
#include <stdio.h>
c_developer()
{ int a=30, b=90;
  printf("%d", a+b);
}
Void main()
{ printf("main begins\n");
  c_developer();
  printf("main ends\n");
}
```

O/P:

```
main begins
120 main ends
```

Func. is executed, with the help of a func. call only inside main().

- empty-case:

```
c_developer()  // called func. or func. def.
{ printf("hi");
}
main()
{ c_developer(68,1); // calling func. or func. call
}
```

O/P:
hi

→ Here Arg.s of calling func. are discared, and prints stmts in called func. It doesn't throw any errors.

Func. with no return type and args:

```
#include <stdio.h>
c_developer(int a, int b) // parameters or formal arg.s
{ printf("%d\n", a+b);
}
main()
{ c_developer(68,1); // arg.s or actual arg.s
}
```

O/P:
69

(int a, int b) // allowed
(int a,b) // not allowed
(int a, int a) // not allowed

Func. ^call with return type & ~~no~~ args:

In func. ^call with ~~no~~ arg., only control is transferred to called func, ~~whereas func call with args,~~ even args. are passed.

Calling & Called func.

void is a keyword. It is a return type, where nothing is returned.

→ When return type is void, if we try to use return stmt., then only transfer not data.

```
void ___ ()
{return; //allowed
 return ___; //not allowed
}
```

→ We use other data-types for storing data, acc. to req.

```
#include <stdio.h>
int c-developer (); //func. dec.

int c_developer () //func. def.
{ int a=6, b=9;
  int res=a*b;
  return res;
}
```

```
Void main()
{ int res = c_developer();
  printf("%d", res);
}
```

O/P:
54

```
#include <stdio.h>

int c_developer(int a, int b)
{ return a*b;
}

Void main()
{ int a, b;
  printf("Enter a:"); scanf("%d", &a);
  printf("Enter b:"); scanf("%d", &b);
  int res = c_developer(a,b);
  printf("Multiplication of 2 no: %d \n", res);
}
```

Q1. Last-digit of a no. : n % 10 = result

Q2. count no. of digits ?

Division by 10

$\Rightarrow n/10$

$154 \div 10 = 15$

$15 \div 10 = 1$

$1 \div 10 = 0$

count = 3

```
count=0;
count++;   1
++;        2
++;        3
```

Code):

```
count=0;
while(n!=0)
{ n /=10;
  count++;
}
return count;
```

Q3. sum of Digits of a no.

154/10 ⇒ 15
154 % 10 ⇒ 4
(Tracing IMP (Here))

```
main()
{ int n, sum=0, dig=0;

printf("");
scanf("%d", &n);

while(n!=0)
{ dig = n%10; // Extract Last Digit
  sum += dig; // Adding
  n /=10; // Removing Last Digit
}
printf("%d\n", sum);
}
```

Steps:

Repeat it until n==0:

1. Extract Last Digit into a Var.
2. Adding Last Digits into a Var.
3. Removing Last Digit (/=10)

Page No.
Date / 120

Q. Factors of a No:

1). 1 to N ⇒ traverse
2). Then, check Remainder is 0
3). If Yes, print it

```
for (i=1; i<=n; i++) //2
{  if (n%i==0) //2
       print("%d ", i); //3
}
```

eg:
n=6

	i=1		
	1<=6	6%1 ✓	1
i=2	2<=6	6%2 ✓	2
=3	3<=6	3%6 ✓	3
=4	4<=6	4%6 != 0	
=5	<=6	5%6 != 0	
=6	<=6	6%6 ✓	6

O/P:
1 2 3 6

** Don't Memorize the Programs **

✓ (1). Und. the steps
(2). Recall / Teach

Q. Factorial of a given no.:

```
fact=1;
for (i=1; i<=n; i++)
    fact *= i;
return fact.
```

Q- Sum of Factorial of each digit

3.

1! + 2! + 3!

1 + 2 + 6 = 9

~~Q Sum of Fac~~

1. Extract Last Digit
2. Factorial of it
3. Sum of Factorials
4. Remove Last Digit

```
#include <stdio.h>
void main()
{ int n, dig=0, fact=1, sum=0;
  printf("\n");
  scanf("%d", &n);

  while(n!=0)
  { dig = n%10;
    fact = dig * fact;
    sum = sum + fact
    n = n/10;
  }
  printf("%d", sum);
}
```

Page No.
Date / /20

Method-1:

```
#include<stdio.h>
int factorial (int n)
{ int i, fact=1;
  for (i=1; i<=n; i++)
      fact = fact * i;
  return fact;
}

Void main()
{ int  n, fact=1, sum=0, dig=0, i;
  print ("  "); scanf ("%d", &n);
                                  strong
                                  x = n
  while (n!=0)
  { dig = n%10; //1
    fact = factorial (dig); //2
    sum += fact; //3
    n /=10; //4
  }

                                  strong
                                  If (x == n)
print ("%d\n", sum);              else printF(strong)
                                       printf (Non);
}
```

OR

```
(No func. req.)
 while (num!=0)
{ dig = num%10; //1
  fact=1;  //2
  for (i=1; i<=digit; i++)
  { fact = fact * i;
  }
   sum+=fact;
   num/=10;
}
```

Sum of factorials:

```
                i<=n
for (i=1; ~~i<=n~~; i++)
{  fact = fact * i;
   sum = sum + fact;
}

printf ("%d \n", sum);
```

Q. Reverse a No.

123 ⇒ 321

Extract Last Digit -

Remove

```
  3
12
```

sum = sum + n%10

```
#include <stdio.h>
{ int
  printf ("  "); scanf ("%d", &n);
                                x=n
 while(n!=0)
 { dig = n%10;
   rev = rev*10 + dig;
   n=n/10;
 }
 printf ("%d", rev);
}
```

if (x == rev)
else (Palindrome
(non)

Page No.
Date / /20

I/P:

256

n	dig	rev	rev=(rev*10+dig);
256	6	0	
25	5	6	
2	2	65	
0		652	

Perfect No:

eg: 6

Factors: 1+2+3 (6 not considered)
= 6

X 8

factors: 1+2+4
= 7 ∝

```
#include <stdio.h>
void main()
{ int n,i, prev_fact=0;
  printf("n"); scanf("n", &n);

  for(i=1; i<n; i++)
  { if(n%i==0)
      prev_fact += i;
  }

  if(prev_fact==n)
     printf(perfect)
  else
     printf(non);
}
```

Strong No.:

eg: 145 ∈ (V) some

$1! + 4! + 5!$

$1 + 24 + 120 = 145$

eg: 123 ∉ (α)

$1! + 2! + 3!$ (α)

$1 + 2 + 6 = 9$

Q. Given no. is prime or not.

(Ans).

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

void main()
{ int n,i;
  printf(" "); scanf("%d",&n);

  if (n==1 || n==2)
      printf("Prime Number \n");

  for (i=2; i<=sqrt(n); i++)
  { if (n%i==0)
    { printf("Non Prime Number \n");
      exit(0);
    }
  }

  printf("Prime Number \n");
}
```

Page No.
Date / /20

Q. prime no.'s b/w 1 to 50.

*** Armstrong No:

1. Count no. of digits
2. Compute the pow(Digit, count) for each digit in no.
3. Add these results.

eg: 143 (✗) (✗)

count = 3

$1^3 + 4^3 + 3^3 = 1 + 64 + 27 = 92$

153 ⇒ (V)

count = 3

$1^3 + 5^3 + 3^3$

$1 + 125 + 27$

$= \boxed{153}$

int su