DU 14/09/19

UNIT-1

(i) **Data**:- It is a raw fact.
E.g. 12, RA, etc

(*) **Information**:-
→ After processed the data will be treated as information.
→ The input to the computer is the data and the output from the computer is the information.

E.g. Age = 12
Name = RA

1st Generation of Comp.	2nd Gen. of Comp.	3rd Gen. of Comp.	4th Gen.
Transitors & tubes	Transitors	Integrated device	~~Integrated device~~ Microprocessor

Scanned by CamScanner

5th Generation:

AI, Robotics, Neural Network

Already 75% developed
- Think like human Being
- Act like " "
- Act " rationally
- Think " "

Intelligent - Processing the Knowledge

⊛ Computer Language and its Generations:

→ The language understood by the computer is called computer language.

→ The binary language or machine level language (0, 1) is known as computer language.

Generations of Computer Language:

1) First Generation (Machine Level)
2) Second Generation (Assembly Level)
3) Third Generation (Middle Level)
4) Forth Generation (High Level)

Scanned by CamScanner

1) First Generation (Machine Level):

→ It is otherwise known as machine level language.

→ This language can be written in terms of 0 and 1.

→ It is machine dependent that is machine can understand by the language directly.

→ It is usual independent that it is very difficult for the user to write and understand.

2) Second Generation (Assembly Level):

→ This language can be written in terms of Mnemonics or Symbolic words.

E.g. ADD, MUL, DIV, etc.

→ It is machine independent language that is a system software is required to convert assembly language to machine language that is named as Assembler.

3) Third Generation (Middle Level Language):

→ This is otherwise known as middle level language.

→ This language is in between assembly level and high level languages that is by using this language the compilers of high level

Scanned by CamScanner

Languages are written.

E.g. C, C++, etc.

→ It is machine independent that is this languages can be written by using english alphabets.

→ Two system software are required to convert middle level language to machine level language.

E.g. Compiler, Interpereter

→ A Compiler converts the middle level language into machine level as a whole at a time.

E.g. C, C++, etc. are the compiler based language.

An Interpreter converts the middle level language into machine level line by line.

E.g. LISP, PROLOG, JAVA, etc.

4) Forth Generation (High Level Language):

→ This language can be written by using english alphabets.

→ It is machine Independent and User dependent.

E.g. JAVA, Oracle, SQL, etc.

Scanned by CamScanner

Algorithm:-

→ An algorithm is a step by step procedure to write the programming languages.

→ In algorithm, it must contain One Start and One End state.

→ E.g.2) Write an algorithm to find out sum of two numbers.

Ans- Step 1: Start

Step 2: Input A, B

Step 3: Sum = A+B

Step 4: Display Sum

Step 5: End

Q) Write an algorithm to find out greatest among two numbers.

Ans- Step 1: Start

Step 2: Input A, B

Step 3: If A is greater than B then

display "A is greater"

Otherwise

display "B is greater"

Step 4: End

Scanned by CamScanner

Q) Write an algorithm to find out area of a triangle.

(Area $= \sqrt{s(s-a)(s-b)(s-c)}$)

Ans. Step 1: Start

Step 2: Input s, a, b, c

Step 3: Area $= \sqrt{s(s-a)(s-b)(s-c)}$

Step 4: Display Area

Step 5: End

Q) Write an algorithm to find out largest among 3 numbers.

Ans Step 1: Start

Step 2: Input a, b, c

Step 3: If (a > b) and (a > c)

display "a is greater"

Otherwise

if (b > a) and (b > c)

display "b is greater"

Otherwise

display "c is greater"

Step 4: End

Scanned by CamScanner

Q) Write an algorithm to find out sum of 10 input numbers

Ans Step 1: Start

Step 2: C = 1, Sum = 0

Step 3: Input A

Step 4: Sum = Sum + A

Step 5: C = C + 1

Step 6: If (C <= 10), then go to

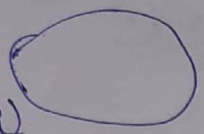
Scanned by CamScanner

Dt: 16/9/19

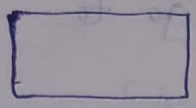
⊛ Flow Chart:

It is a pictorial representation or diagramatic representation

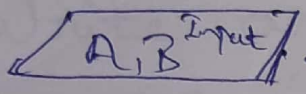
→ It consists of set of symbols such as



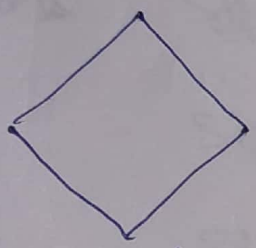
(oval) → Start / End



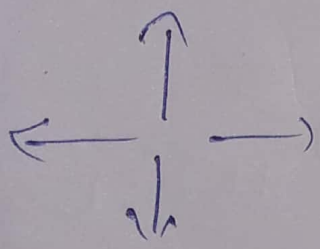
(rectangle) → Processing



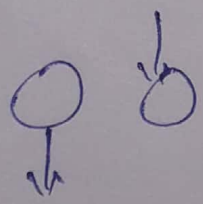
(parallelogram) → Input / Output



(rhombus) → Condition



→ Flow Lines

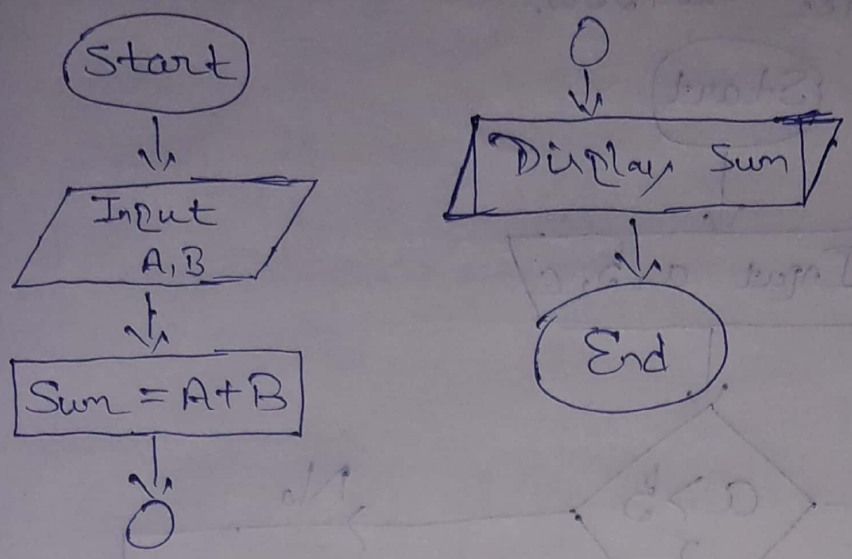


→ Connectors

Scanned by CamScanner

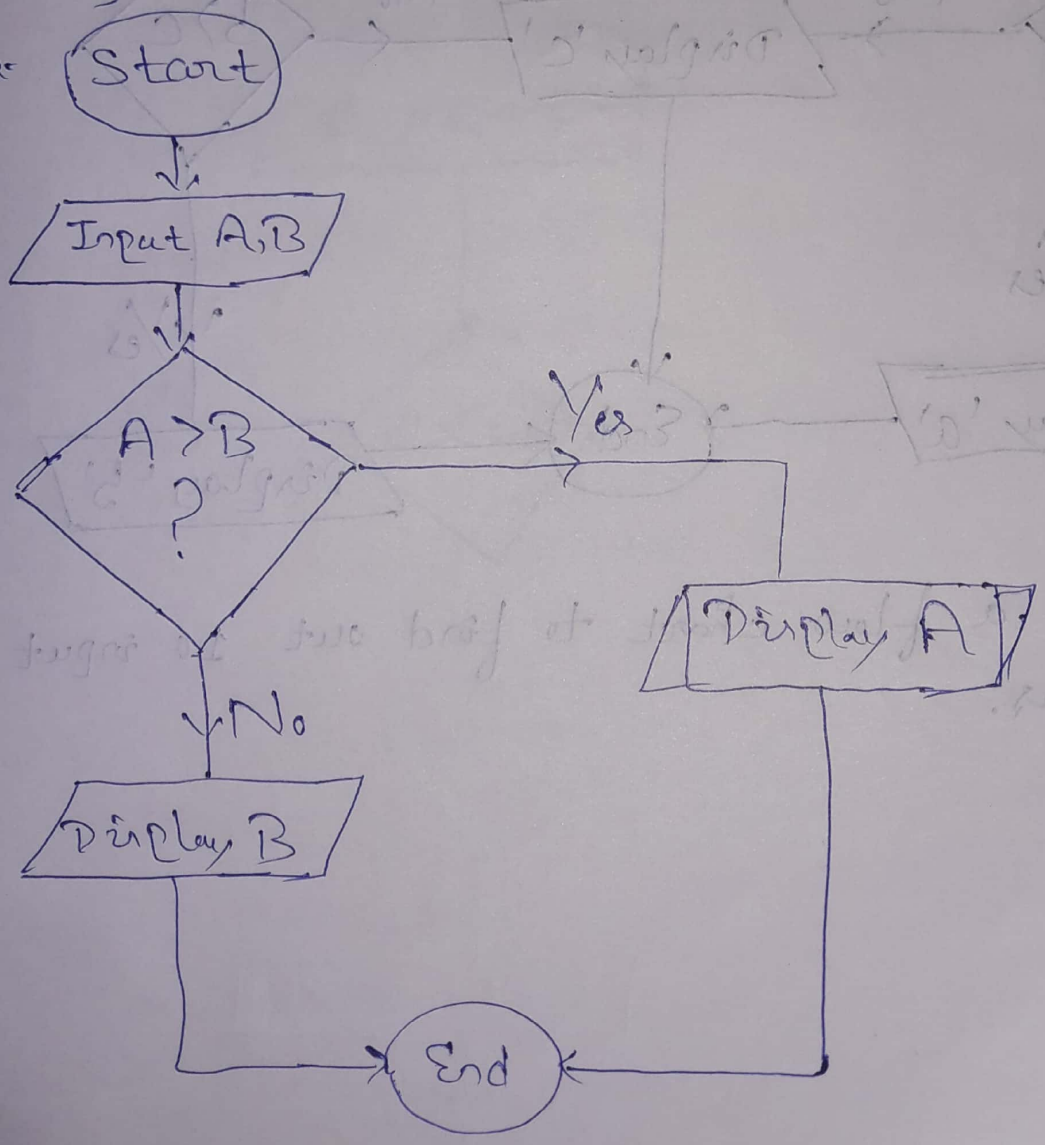
Q) Draw a flow chart to find out sum of two numbers.

Ans-



Q) Draw a flow chart to find out greatest among two numbers.

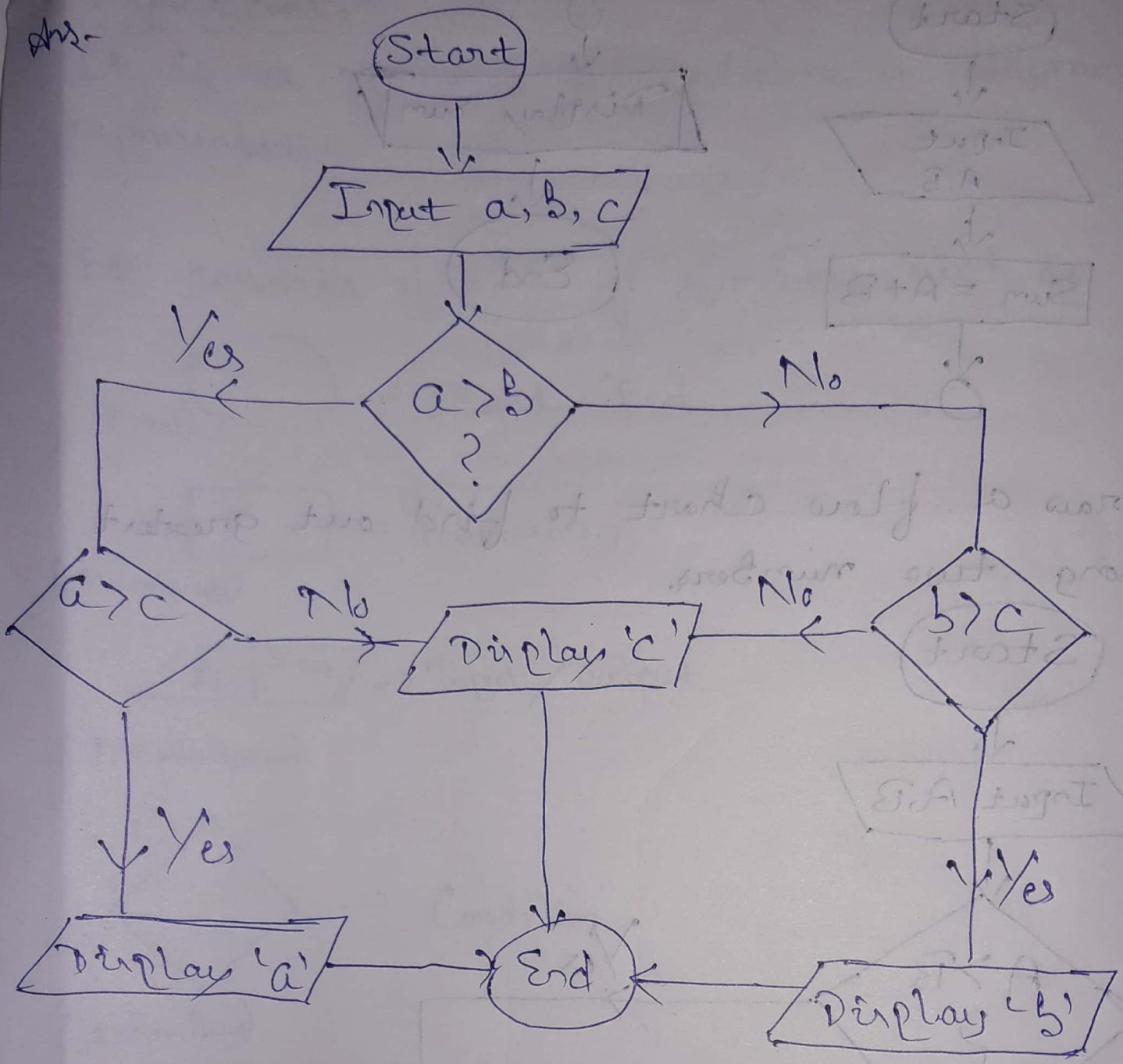
Ans-



Scanned by CamScanner

2) Draw a flow chart to find out greatest among three numbers.

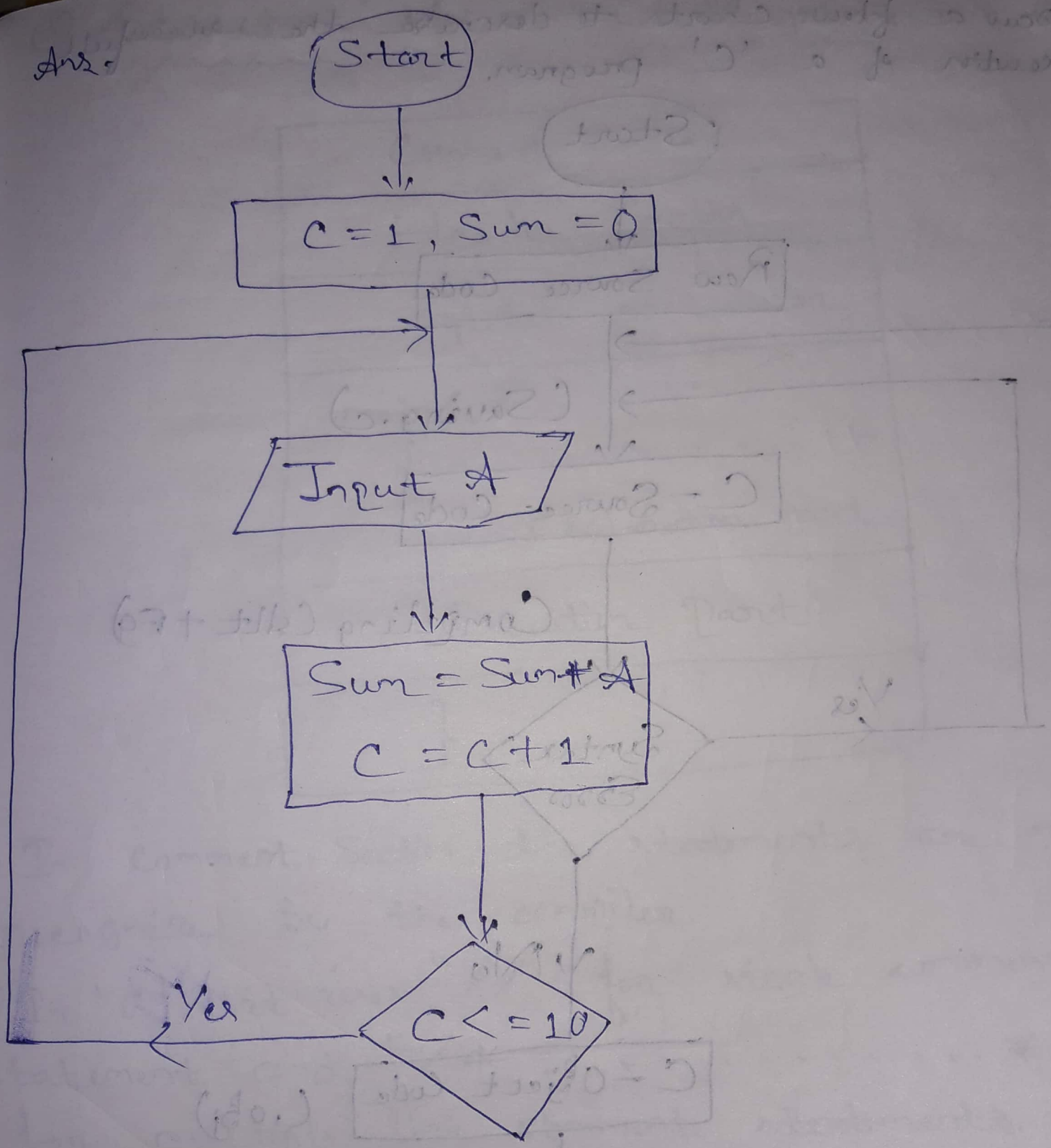
Ans-



2) Draw a flow chart to find out 10 input numbers.

Scanned by CamScanner

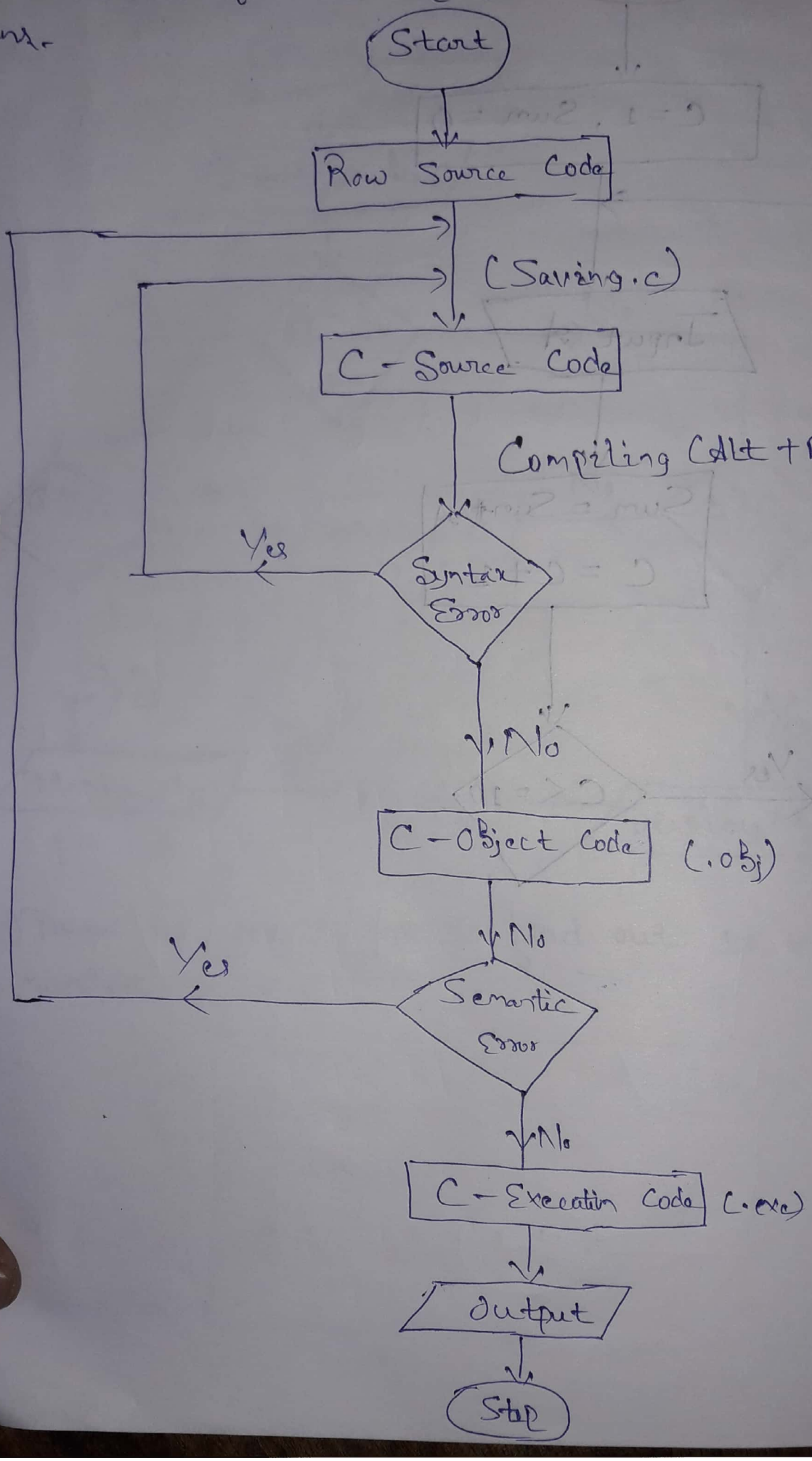
Ans.



Scanned by CamScanner

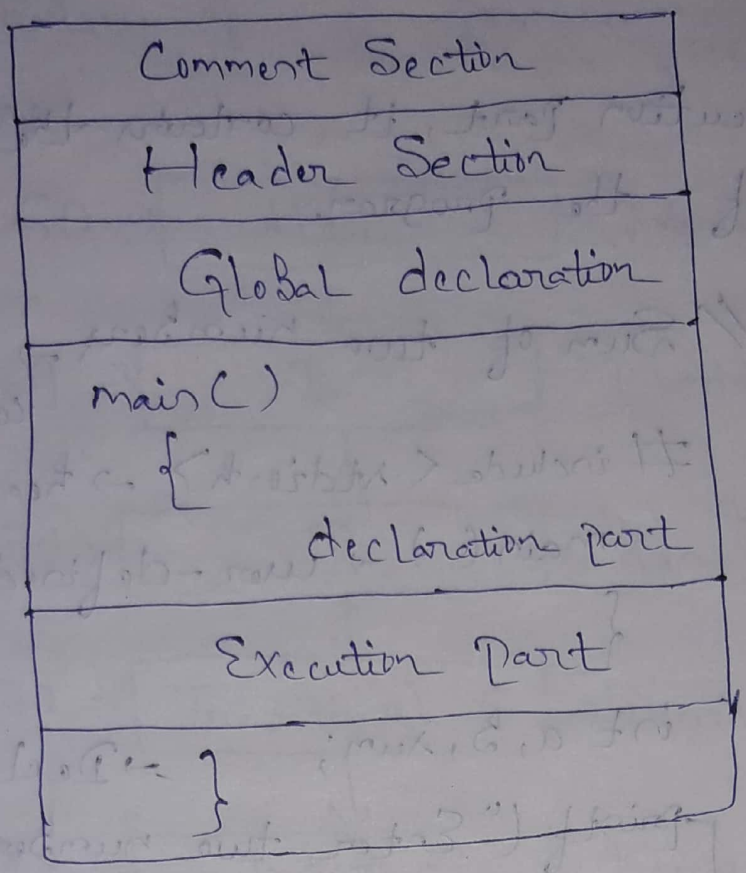
Q) Draw a flow chart to describe the successfull execution of a 'C' program.

Ans.-



Scanned by CamScanner

⊛ Structure of a 'C' Program:



1) In Comment Section, the statements are not recognize by the compiler.

2) In 'C', it uses "//" for single comment statement and "/* ..... */" for multiple line comment statements.

3) In header section, the user can define header files that are required by the programmer.

4) In Global declaration section, the user can declare global variables.

5) "main()" is a user-defined function and it indicates the starting point of the programming.

Scanned by CamScanner

6) In declaration part, the user declare all the variables that are used in the execution part.

7) In execution part, it contains the whole logic of the program.

E.g.

```
// Sum of two numbers →  comment section
#include <stdio.h> → header section
main ( ) → user-defined fun
{
int a, b, sum;          → Declaration part
printf ("Enter two numbers:");   Library fun
scanf ("%d%d", &a, &b);
sum = a + b;
printf ("The Sum is %d", sum);
}
```

Execution part

Output

```
Enter two numbers:
5
6
The Sum is 11
```

Scanned by CamScanner

⊛ History of 'C' Program

Dt. 19/9/19

→ The history of 'C' programming can be explained as

1960 → ALgol → International Group

↓

1969 → BCPL → Martin Richard

↓

1970 → B → Ken Thomson

↓

1972 → Traditional C → Denis Ritchie

↓

1978 → K&R C → Keing & Ritchie

1989 → ANSI C → ANSII Committee

1990 → ISO C → ISO Committee

1999 → C 99 → Standarization Committee

→ 'C' programming language was initially developed by the scientist Denis Ritchie at Bell Laboratories in 1972.

⊛ Features of 'C' or Importance of 'C':

1) 'C' is a middle level language.

2) 'C' is highly portable that is 'C' programs

Scanned by CamScanner

written in one computer can be run in another computer with no modification, or with little modification.

3) 'C' is ability to extend to itself that is 'C' programming is barically a collection of vast library functions and besides that we can add some other user-defined functions.

4) 'C' is a rubust language that is 'C' contains a rich set of library functions and operators that can be used to write a program for any type of complex problem.

5) 'C' is simple that is it contains structural or procedural approach which is easy for a compiler as well as for the programmer.

⊛ Tokens in C:

→ A ~~tok~~ token can be defined as a smallest word in a sentence and in 'C' the tokens can be classified in the following categories:

1) Operators
2) Identifiers
3) Keywords
4) Constants
5) Strings
6) Data types
7) Special Symbols

Scanned by CamScanner

1) Operators:

→ In 'C' it supports the following operators:

i) Arithmetic Operator

ii) Relational Operator

iii) Logical Operator

iv) Conditional Operator

v) Bitwise Operator

vi) Assignment Operator

vii) Increment/Decrement Operator

viii) Special Operator

i) Arithmetic Operator:

→ These operators are used in a arithmetic expression and in 'C' it supports the following arithmetic operators:

+ (Addition)

− (Substraction)

* (Multiplication)

/ (Division)

% (Modulo division)

E.g. −5 / −2 = 2

Scanned by CamScanner

$-5 \% -2 = -1$
$5 \% -2 = 1$
$-5 \% 2 = -1$

Rule - In case of modulo division, it ignores the sign of the operands during the division process and finally it will put the sign of the first operand to the result.

* Precedence and Associativity of arithmetic operators:

→ In 'C' the precedence of arithmetic operators can be defined as:

1) Division, modulo division, multiplication
2) Addition, Subtraction

→ If more than one operators in an expression are belongs to same precedence level than we can apply a technique called "associativity" of the operators that is the ~~expression~~ operators will be evaluated in the same precedence level in FCFS (First Come First Serve Basis).

Scanned by CamScanner

a) Evaluate

$5 * 4 / 2 - 6 * 3 + 9$

$= 20 / 2 - 6 * 3 + 9$

$= 10 - 6 * 3 + 9$

$= 10 - 18 + 9$

$= -8 + 9$

$= 1$

a) Evaluate, $5 * 4 * 3 * 2 * 6$

$= 20 * 3 * 2 * 6$

$= 60 * 2 * 6$

$= 120 * 6$

$= 720$

a) Evaluate, $5 + 4 * 2 / 3 - (6 \% 4 + 8)$

$= 5 + 4 * 2 / 3 - (2 + 8)$

$= 5 + 4 * 2 / 3 - 10$

$= 5 + 8 / 3 - 10$

$= 5 + 2 - 10$

$= 7 - 10$

$= -3$

Scanned by CamScanner

2) Relational Operator:

→ These operators are used to compare between two operands.

→ The Output of a relational operator is either True (1) or False (0).

→ In 'C' it supports the following relational operators:

< (less than)

<= (less than or equal to)

> (Greater than)

>= (Greater than or equal to)

== (Comparison operator)

!= (Not equal to)

E.g.

```
main( )
{
int a=5, b=7, c;
c = a<=b;
printf("%d", c);
}
```

Output : 1

Scanned by CamScanner

③ Logical Operators:

→ These operators can be used in between two relational expressions.

→ Like relational operators, logical operators also return either 0 or 1.

→ In 'C', it supports the following logical operators:

&& (logical AND)
|| (logical OR)
! (logical NOT)

Logical AND (&&)

1 && 1 = 1
1 && 0 = 0
0 && 1 = 0
0 && 0 = 0

Logical OR (||)

1 || 1 = 1
1 || 0 = 1
0 || 1 = 1
0 || 0 = 0

Logical NOT (!)

! 1 = 0
! 0 = 1

Scanned by CamScanner

E.g.

```
main ( )
{
int a=5, b=7, c;
c=(a<=b)&&(a>=b);
printf("%d", c);
}
```

Output : 0

④ Conditional Operator:

→ This operator can be represented as "?:"

→ This operator initiates the concept of "if else" statement.

Syntax:

(Conditional expression) ? True statement : False Statement

E.g.

```
main ( )
{
int x=5, y=7;
(x>y) ? printf("x is greater") :
        ("y is greater");
}
```

Output : y is greater

Scanned by CamScanner

⑤ Bitwise Operator:

→ These operators can be applied on a operand at their bit level.

→ In 'C', it supports the following bitwise operators:

& - Bitwise AND

| - Bitwise OR

<< - Left Shift

>> - Right Shift

^ - Bitwise X-OR

Left Shift (<<):

```
main( )
{
int x = 13, z;
z = x << 2;
printf("%d", z);
}
```

$(13)_{10} = (1101)_2$

00001101

00011010

00110100 =

$(52)_{10}$

Output: 52

Scanned by CamScanner

Right Shift (>>):

$(55)_{10} = (110111)_2$

0 0 1 1 0 1 1 1

0 0 0 1 1 0 1 1

0 0 0 0 1 1 0 1

0 0 0 0 0 1 1 0 $= (6)_{10}$

```
main ( )
{
int x=55, z;
z=x>>3;
Printf("%d", z);
}
```

Output: 6

Dt

* 1) OS is discovered by Microsoft.

UNIX Linux (Ubuntu)

FOSS → Free Open Source Software

Linux (Command Oriented O.S.)

Ubuntu Fedora Redchat

Scanned by CamScanner

C:\Users\ICT>

Root → prompt

Types of Command in DOS:

1) Internal Command
2) External Command

CD.. → This command is used to the previous directory.

e.g. C:\> User> CD..

C:\> → This command is used to go to the root write.

MD → (Make in Directory) → It is used to create a new directory.

Syntax: MD < directory name > ↵

e.g. MD MCA1 ↵

Create multiple directory in a single command → MD M1 M2 M3 M4 M5 ↵

CD (Change directory) → This command is used to change the current working directory.

e.g.

C:\Users\ICT>MD MCA1 ↵

C:\User\ICT> CD MCA1 ↵

C:\User\ICT\MCA1>

Drive Change:

C:\User\ICT>CD

F: ↵

F:\>

Scanned by CamScanner

* Create a new file:

COPY CON → This command is used to create a file.

Syntax: COPY CON <filename>.<extension> ↵

Save a file → [F6] ↵ or [Ctrl] + [Z] ↵

* [TYPE] → This command is used to view the content of the file.

Syntax: TYPE <filename> ↵

e.g. C:\Users\ICT\MCA1> COPY CON TEST ↵

This is 1st Sem

[F6] ↵

C:\Users\ICT\MCA1> Type test ↵

[EDIT] → This command is used to modify the content of a file.

Syntax: EDIT <filename> ↵

Alt, F, S → To Save

Alt, F, X → To exit

[REN] → This command is used to change the name of file.

Syntax: Ren <old name> <new name> ↵

Es. REN test L105 ↵

[DEL] → This command is used to delete the file.

Syntax: Del <file> ↵

E.g. Del L105 ↵

[COPY] → This command is used to copy the file(s) from one location to another location.

Scanned by CamScanner

Syntax: COPY <Source path>\<file name> <destination path> ↵

E.g. C:\Users\ICT\MCA1> COPY L105

C:\Windows\Java ↵

or

C:\> COPY Users\ICT\MCA1\L105

Windows\Java\L105 ↵

Move → This command is used to move files from one location to another location.

Syntax: Move <Source path>\<file name>

<destination path> ↵

e.g. ~~Move~~ C:\> Move Users\ICT\MCA1\L105

Windows\Java\L105 ↵

RD → This command is used to remove the file.

Syntax: RD <directory name> ↵

DIR → This command is used to view the contents of directory.

Syntax: DIR <directory name> <switches> ↵

E.g. C:\Users\ICT\MCA1> dir ↵

C:\>dir C:\Users\ICT\MCA1 ↵

/P → Page wise

/W → Tabular list

Scanned by CamScanner

Wild card Characters:

? → Single Character

* → Multi Character

These characters are used for selecting multiple files.

*sh → names having in sh last position.

E.g. C:\ABC

- → Rajesh
- → akash
- → Bimal
- → anil
- → Ramesh
- → Sunil

Filenames having 'a' in second position.

? a *

Q) Second letter should be 'a', forth letter should be 'e' and it must be a name having five characters.

Ans. ? a ? e ?

Q) Select all file having 4 characters.

Ans. * * * *

Tree → This command is used to view the contents of a directory in a hierarchial manner or tree structure.

Syntax: Tree <directory name> <switches>

E.g. C:\> Tree /F

Scanned by CamScanner

ATTRIB → This command is used to change the files.
attribute

ATTRIB <filename> <switches>

+r → to apply read only attribute
-r → to apply remove only attribute
+h → to apply hidden only attribute
-h → to apply remove only attribute
+s → to apply system file
-s → to apply remove file.

Arithmetic Operator (=)

Symbol	Meaning / Example
=	a=5 a=-3
+=	a=5 a+=3 ⇒ a=a+3 a=8
-=	a=5 a-=3 ⇒ a=a-3 =2
*=	a=5 a*=3 ⇒ a=a*3 =15
/=	a=5 a/=3 ⇒ a=a/3 =5
%=	a=5 a%=3 ⇒ a=a%3 =2

Scanned by CamScanner

⊛ Increment and Decrement Operator:

This operator simply increments the value of a variable.

→ In 'C', it supports "++" as increment operator and "--" as decrement operator.

E.g. i++ indicates i+1

i-- indicates i-1

→ If these operators are used before the operands then these are called pre-increment or decrement and if these are used after the operands, then their called post-increment or decrement.

→ In E.g. i++ (post-increment)
++i (pre-increment)
i-- (post-decrement)
--i (pre-decrement)

First use, then assign

1st assign, then use

→ In case of pre-increment or pre-decrement, first it will perform the operation & then it will assign the operands.

→ In case of post-increment or decrement, first it will assign the value and then perform the ~~assign~~ operation.

Scanned by CamScanner

E.g.
```
main ( )
{
int i=5;
printf ("%d", i++);
printf ("%d", ++i);
}
```

Output: 5 7

→ In an arithmetic expression, first it will perform all the pre operation, then it will perform all the binary operation, assign the value to the left hand side variable & then go for evaluating all the post operation.

e.g.
```
main ( )
{
int i=5, j;
j = i++ + ++i;
printf ("%d%d", i, j);
}
```

Output: 7 12

e.g.
```
main ( )
{
int i=5, j;
j = ++i + i++ + i++ + ++i;
printf ("%d%d", i, j);
}
```

Output: 9 28

Scanned by CamScanner

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

Output: 6 36

```
main ( )
{
int i=9, j;
j = i++ % 4;
printf ("%d%d", i, j);
}
```

Output: 10 1

Special Operators:

→ 'C' supports some special operators such as

i) Comma operator (,)

ii) Size of operator (sizeof)

iii) Pointer Operator (& and *)

iv) Member Selection Operator (., →)

→ The comma operator can be used to leave the related expressions together.

Scanned by CamScanner

e.g. int a;
int b;
int c;
} int a, b, c;

→ The size of operator can be used at compile time and for this it is otherwise known as compile time operator.

This operator returns the size of the operands in terms of bytes during compilation time.

e.g.

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

Output: 2

Scanned by CamScanner

Introduction to Programming Language:

→ Set of instructions to perform a specific task is called program.

Collection of related programs is called Software.

Alphabet → Word → Sentence → Paragraph → Book

Character set → Token → Instruction → Program → Software

(The set of symbol)

- Literals
- Identifier
- Keywords
- Operators

→ 'C' language was discovered by Dennis Ritchie at the Bell Laboratories in 1972.

Languages of Computer:

→ Low level / Machine (binary 0, 1)

→ Assembly

→ High level

0 ⇒ 5V (low)
1 ⇒ 10V (high)

Assembler → To transfer assembly language into low level / machine level language.

Scanned by CamScanner

Compiler, Interpreter → To translate high level language into low level/ machine level language.

For e.g. java

Compiler	Interpreter
1) Converts the whole statement at a time.	1) Converts the statement line by line.
2) It requires more space.	2) It requires less space.
3) It performs the function faster. E.g. C, C++.	3) It performs the function slower. E.g. java, PHP, Python, Visual basic

Software

System Software

→ It is hardware specific.

→ It is used for operating computer hardware.

E.g. Operating System like macOS, Ubuntu, Microsoft Windows, etc

Application Software

→ It is user's specific.

→ It is used by user to perform specific task.

E.g. Game, banking, railway reservation, etc.

Scanned by CamScanner

Identifier:

→ It is the name given to any variable, function, array, structure, union and pointer by which they are identify.

Rules for identifier:

1) Identifier name must contain digits, alphabets and underscore. e.g. a1

2) The first character must be an alphabet or underscore.

e.g. a1 or _a1 ✓

1a ✗

_a ✓

3) No white spaces are allowed in the identifier name.

4) No Keywords are defined as identifier name.

5) It is case sensitive.

E.g. a and A are not same.

Keywords:

→ These words are fixed words and can be understood by the compiler itself.

→ The Keywords are case sensitive

Scanned by CamScanner

e.g. int ✓ INTX for ✓ while ✓

→ In 'C' it contains 32 numbers Keyword.

ASCII → American Standard Code for Information Interchange.

A - 65 (ASCII value)
B - 66
⋮
Z - 90

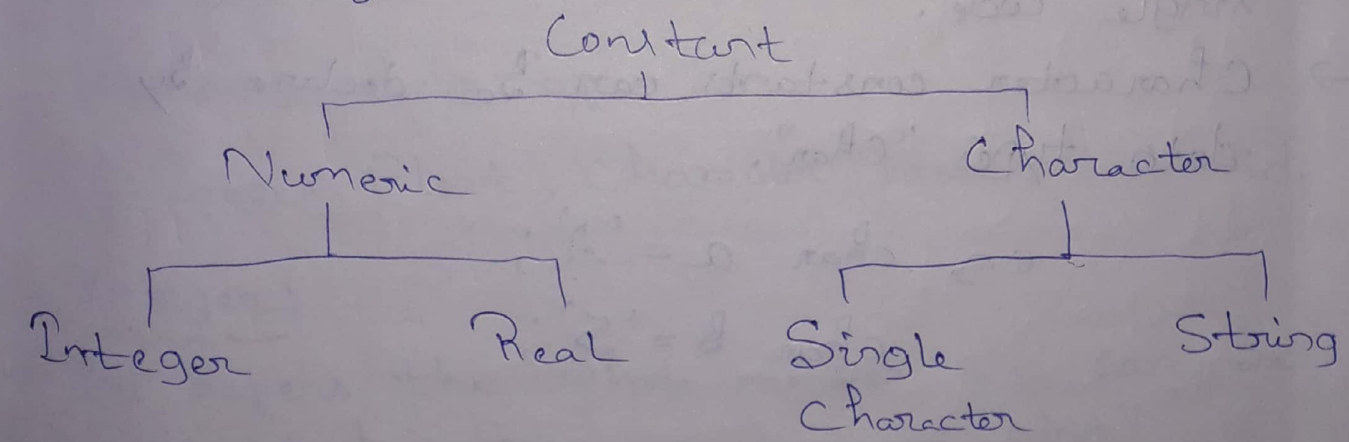
a - 97 (ASCII value)
b - 98
⋮
z - 122

0 → 48 (ASCII value)
1 - 49
⋮
9 - 57

Constants:

→ It refers the fixed values that do not change during the compilation time.

→ In 'C', the content can be categorised in the following manner:



Scanned by CamScanner

Integer Constant:

→ It indicates a single digit or multiple digits and it may be negative or positive number.

→ These constants are declared by using the data type "int".

e.g. int a=5;
int b = 5/2;

Real Constant:

→ It indicates a sequence of digits separated by decimal point.

→ Real constants can be declare by using the data type "float" or "double".

e.g. float a = 5.4;
double b = 9.81;

Single Character Constant:

→ It refers a single character, that may be a digit or alphabets enclosed within single code.

→ Character constants can be declare by data types "char".

e.g. char a = 'A';
char b = '5';

Scanned by CamScanner

String Constant:

→ It refers either a single character or sequence of character enclosed within double codes.

→ String constants can be defined by using the data type "char".

e.g. char a[ ] = "A";

char a[ ] = "welcome";

Data types:

→ It indicates the type of a variable and in 'C', the data types can be clarified into 3 categories:-

1) Primitive data type
2) Derrived data type
3) User-defined data type

1) Primitive data type:

→ These data types are otherwise known as fundamental data types and it includes the following data types:-

Integer, Float, Character, Double and void.

Integer:

→ It refers the whole number and can be declared by using a keyword "int".

→ The integer may be a signed or unsigned.

Scanned by CamScanner

→ The size of an integer variable is 2 Byte and its range can be defined as

-2^{15} to $2^{15}-1$.

e.g. int a, b;

→ The integer data can also be defined as "long int" and its size is 4 Bytes and its range can be defined as -2^{31} to $2^{31}-1$.

e.g. long int a;

Float:

→ It refers the sequence of digits separated by a decimal point & can be declared by using the keyword "float".

The size of a floating variable is 4 Bytes and its range can be de defined as -2^{31} to $2^{31}-1$.

e.g. float a;

Character:

→ It refers a single character that is enclosed within a single code.

→ It can be declared by using a keyword "char".

→ The size of character variable is 1 Byte and its range can be defined as

-2^{7} to $2^{7}-1$.

e.g. char a;

Scanned by CamScanner

Double:

→ It refers the floating number but its size is greater than the ~~following~~ floating number.

→ It can be declared by using a keyword "double".

→ The size of double is 8 bytes and its range can be defined as -2^{63} to $2^{63}-1$.

e.g. double a;

void:

→ This data types are no value and these are usually used to specify the type of the function i.e. If a function is defined by using void data type, then it does not give any return values.

```
e.g. void main()        int main()
     {                  {
     ____               ____
     ____               ____
     ____               ____
     }                  }
```

(with void) (without void)

Scanned by CamScanner

Dt: 24/10/19

Conditional Statements:

In 'C', it supports the following conditional statements:

1) If statement
2) switch statement
3) go to statement
4) conditional operator statement

1) If statement:

if statements are again classified into 3 categories:

i) simple if statement
ii) If else statement
iii) if-else-if statement

i) simple if statement:

syntax:

```
if (test condition)
    true statement;
```

or

```
if (test condition)
  {
    statement 1;
    statement 2;
    ...
  }
```

Scanned by CamScanner

```
void main( )
{
 int i=5, j=6;
 if (i<5)
 printf ("Hello");
}
```

Output: No output

```
void main ( )
{
 int i=5, j=6;
 if (i<j)
 printf ("Hello");
 printf ("Hii");
 printf ("Hello");
}
```

Output: Hii, Hello

ii) if else statement:

Syntax:

```
if (test condition)
        true statement;
else
        false statement;
```

Scanned by CamScanner

e.g.

```
void main( )
{
int i=5, j=6;
if (i<j)
printf ("Hello");
else
    printf ("Hii");
}
```

Output : Hello

iii) if-else-if statement:

Syntax:

```
if (test condition)
    true statement;
else if (test condition)
    true statement;
else
    false statement;
```

e.g.

```
void main( )
{
int i=5, j=6;
if (i<j)
```

Scanned by CamScanner

```
    printf ("Hello");
else if (j < i);
    printf ("Hii");
else
    printf ("FMV");
}
```

Output: Hello

Q) What is nested if statement?

It indicates there are many if statements in a nested manner and the syntax of nested if can be described as:

```
if (test condition)
{
    if (test condition)
    {
        true statement
    }

}
```

e.g.

```
void main()
{
    int i=5, j=6, k=7;
    if (i<j)
```

Scanned by CamScanner

```
{
  if (j<k)
    printf ("Hello");
  }
}
```

Output: Hello

If else ladder / Dangling if else statement

~~If else~~ The concept of if else ladder is there are more than one if-else-if statement in a nested manner i.e. if else ladder.

At the time of compiling if else ladder or nested if else if statement the compiler may be in a dangling situation which is known as dangling if-else-if statement and to avoid this situation 'C' supports another conditional statement i.e. "switch statement".

2) WAP in C to find out greatest among three numbers.

Scanned by CamScanner

Ans.

```
void main( )
{
 int a, b, c;
 printf ("Enter three nos.:\n");
 scanf ("%d%d%d", &a, &b, &c);
 if((a>b) && (a>c))
  printf ("%d is greater", a);
 else if ((b>a) && (b>c))
   printf ("%d is greater", b);
 else
  printf ("%d is greater", c);
}
```

Output:

```
Enter three nos.:
5
15
45

45 is greater
```

② Switch statement:

```
switch (Expression value)
{
 case value-1: Statement1;
               break;
 case value-2: statement2;
               break;
```

Scanned by CamScanner

— — — —

— — — — —

default: default statement;
break;
}

→ Here, the expression value must be either an integer constant or character constant.

→ The value-1, value-2, ... are also either integer constant or character constant.

→ Here, the expression value will match with the case values and ~~any~~ if any matching is found then, the particular case statement will be executed otherwise default statement will be executed.

→ Case statement must be ended with a colon.

→ 'break' statement is optional at each case statement.

→ 'default' statement can be divide anywhere inside the switch block but it is prefer at the end.

Scanned by CamScanner

e.g.

```
void main( )
{
  int n=2;
  switch (n)
  {
  Case 1: printf ("Hello");
          break;
  Case 2: printf ("Hiii");
  Case 3: printf ("FMU");
          break;
  default: printf ("Welcome");
  }
}
```

Output: Hiii FMU

3) goto statement:

⇒ This statement is an unconditional branching statement and to perform this 'C' supports a keyword "go to".

```
goto Label_name;
_ _ _ _ _
_ _ _ _ _ _
_ _ _ _ _ _
Label_name: _ _ _
```

Label_name:

Scanned by CamScanner

Here, the label name must be a valid identifier name.

-) Here, the label name either on the top of the goto statement or at the buttom of the goto statement.

e.g.

```
void main( )
{
int i=5, j=6;
goto FMU;
if (i<j)
printf ("Hello");
else
printf ("Hii");
FMU: printf ("Welcome");
```

Output: Welcome

```
void main( )
{
int i=5, j=6;
if (i<j);
printf ("Hello");
else
printf ("Hii");
printf ("Welcome");
}
```

O/P
Welcome

Scanned by CamScanner

Conditional Statement:

This operator can be represented as "? :".

→ This operator can be acting as "if-else" statement.

Syntax:

```
(test expression) ? true statement : false statement
```

```
void main()
{
int i=5, j=6;
(i<j) ? printf("Hello") : printf("Hi");
}
```

Output: Hello

Looping and Branching Statement:

In 'C', it supports the following looping statements:

1) while statement
2) do-while statement
3) for statement

While statement:

Syntax:

```
initialization of variable;
while (test condition)
{
```

Scanned by CamScanner

```
Statement 1;
Statement 2;
- - - - - -
Re-initialization of variable;
}
```

e.g.

```
void main( )
{
int i=5;
while (i>0)
{
printf ("Hello");
i--;
}
}
```

Output: Hello - 5 times

2) do-while statement:

syntax:

```
initialization of variable;
do
{
statement 1;
Statement 2;
- - - - - -
Re-initialization of variable;
} while (test condition);
```

Scanned by CamScanner

e.g.
```
void main( )
{
int i=5;
do
{
printf ("Hello");
i--;
} while (i>0);
}
```

Output: Hello Hello Hello Hello Hello

3) for statement:

```
for ( initialization of variable ; test condition ; Re-initialization of variable )
{
statement 1;
statement 2;
- - - -
- - - -
}
```

e.g.
```
void main ( )
{
int i=5;
for( ; i>0; i--)
{
printf ("Hello");
} }
```

Scanned by CamScanner

Output: Hello 5 times

```
1) for (i=5; ; )
   {
     printf ("Hello");
   }
```

O/P
Hello infinity

```
2) for (i=5; ; i--)
   {
     printf ("Hello");
   }
```

O/P:
Hello 5 times

WAP to input a number and find out the sum of given L

Logic

(e.g. 123 = 1+2+3 = 6) sum = 0

1234 ans

123/10 = 12
i = 123%10 = 3

12/10 = 1
12%10 = 2

1/10 = 0
1%10 = 1

Ans.
```
void main ( )
{
int sum=0, n, r;
```

Scanned by CamScanner

```
printf ("Enter a number::\n");
scanf ("%d", &n);
while (n>0)
  {
  r=n%10;
  sum=sum+r;
  n=n/10;
  }
printf ("The sum of digits is::%d", sum);
}
```

Output:

```
Enter a number::
1 2 3
The sum of digits is::
6
```

2) WAP in 'C' to input a number and find out the reverse of a number.

Ans.

```
void main ()
{
int sum=0, n, r;
printf ("Enter a number::\n");
scanf ("%d", &n);
while (n>0)
```

Scanned by CamScanner

```
{
r = n % 10;
sum = sum * 10 + r;
n = n / 10;
}
printf ("The reverse of no. is :: %d", sum);
}
```

Output:

```
Enter a no. ::
1 2 3
The reverse of no. is ::
3 2 1
```

Q) WAP in C to input a no. and check whether whether it is Pallindrome or not.

Ans "PALINDROM" : If the reverse of a number is the original no. then it is called Palindrom.

```
void main ( )
{
int sum = 0, r, n, m;
printf ("Enter a number :: \n");
scanf ("%d", &n);
m = n;
while (n > 0)
```

Scanned by CamScanner

```
{
r = n % 10;
sum = sum * 10 + r;
n = n / 10;
}
if (sum == m)
printf ("PALINDROM");
else
printf ("NOT A PALINDROM");
}
```

Output:

```
Enter a number::
121
PALINDROM
```

Q) WAP to input a number and check whether a number is armstrong or not.

Ans-

```
void main ()
{
int sum=0, n, r, m;
printf ("Enter a number::\n");
scanf ("%d", &n);
m=n;
while (n>0)
{
r=n%10;
sum=sum+r*r*r;
n=n/10;
}
if (sum==n)
printf ("ARMSTRONG");
else
```

Scanned by CamScanner

```
printf ("NOT A ARMSTRONG");
}
```

OUTPUT:

```
Enter a number::
   153
ARMSTRONG
```

Q) WAP in 'C' to find out the first 10 fibonacci numbers.

Ans. Fibonacci number:

The sum of previous two numbers is the current number

```
void main( )
{
int a=0, b=1, c, i;
printf("%5d", a);
printf("%5d", b);
  for ( i=3; i<=10; i++)
    {
      c=a+b;
      printf("%5d", c);
      a=b;
      b=c;
    }
}
```

Scanned by CamScanner

Output:

0	1	1	2	3	5	8	13	21	34

Array:

→ Array variable can contain more than one values of same data type.

→ Array is a derived data types.

→ Array is a ~~static~~ ~~data~~ concept of memory allocation.

→ Array can be classified into 3 categories:

1) Single Dimentional
2) Two Dimentional
3) Multidimensional

1) Single Dimensional:

→ In single dimensional, the array is having only one dimension.

a[0]	a[1]	a[2]	.....	a[9]
5	15	7	.....	20

a

Here, a = names of array

size of array = 10

0, 1, ..., 9 = Indexes of the array

5, 15, 7, ..., 20 = Elements of the array

Scanned by CamScanner

Note:

* The array can be declared as

Syntax: `type array_name [size];`

e.g. `int a[10];`

Initialization of array:

→ Array can be initialized in two ways:

1) Compile Time Initialization

2) Run time Initialization

1) Compile Time Initialization:

In this process of initialization, the array can be initialized in the source program.

e.g. `int a[5] = {15, 20, 7, 9, 14};`

or

`int a[ ] = {5, 20, 7, 9, 14};`

Note:

Here, the array indices will be starting from '0' onwards i.e. here,

14	a[4]
9	a[3]
7	a[2]
20	a[1]
15	a[0]

Scanned by CamScanner

Run-Time Initialization:

In this process of initialization, the array can be initialized at run time and for run-time initialization, the array can be declared as:

e.g. int a[10];

Note:

- In run-time initialization, the size of the array must be mentioned.

e.g. int a[]; // Error

2) WAP to input ten numbers and print them in a reverse order.

Ans.
```
void main ( )
{
int a[20],i;
printf ("Enter the elements of array::\n");
for (i=0; i<=9; i++)
{
scanf ("%d", &a[i]);
}
printf ("The reverse order elements are::
\n");
for (i=9; i>=0; i--)
{
printf ("%5d", a[i]);
```

Scanned by CamScanner

```
    }
}
```

Output:

```
Enter the elements of array::
5
15
20
10
12
14
2
7
1
16

The reverse order elements are::
16 1 7 2 14 12 10 20 15 5
```

2) WAP in C to find out the greatest among input numbers.

```
void main( )
{
int a[50], n, lar, i;
printf("Enter the size of Array::\n");
scanf("%d", &n);
printf("Enter the elements of Array::\n");
```

Scanned by CamScanner

```
for (i=0; i<n; i++)
    {
     scanf ("%d", &a[i]);
    }
lar = a[0];
 for (i=1; i<n; i++)
        {
         if (lar < a[i])
           lar = a[i];
        }
printf ("The Largest no. is :: %d", lar);
}
```

```
Enter the size of Array::
 5
Enter the elements of Array::
20  19  72  7  28
The largest element is::
 72
```

Q) WAP in 'C' to find out smallest among n^{th} number.

Ans.

Scanned by CamScanner

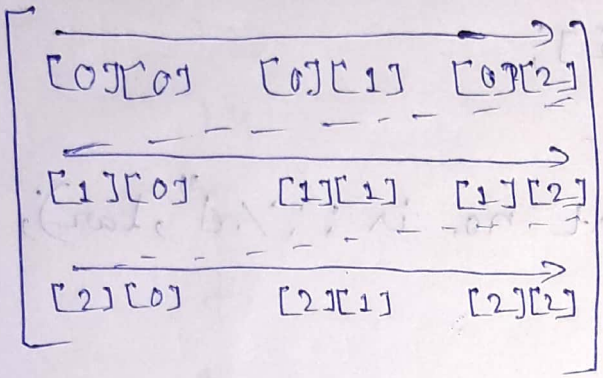
29/10/19

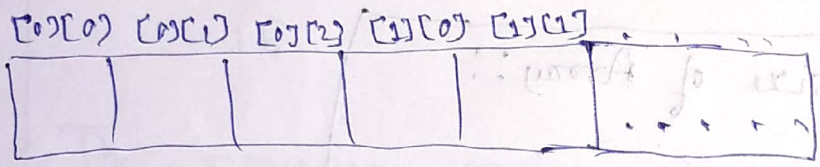
Two-Dimensional Array:-

→ In this concept, the elements of an array can be stored in terms of two-dimension i.e. rows and columns.

→ In Two-dimensional array, the elements are arranged in the memory either in row-major or in column-major order.

Row-major:





Column-major:-

[0][0]

[0][0]

[2][0]

Scanned by CamScanner

→ In row-major, the array elements can be stored in memory on the basis of rows and in column-major the elements can be stored on the basis of columns.

→ The two-dimensional array can be declared as

Syntax: `type array_name [Row_size][Col_size];`

e.g. int a[2][3];

(*) Initialization of 2-D array:

→ The array can be initialized in two ways:

1) Compile-time initialization

2) Run-time initialization

1) Compile-Time Initialization:

e.g. int a[2][3] = {
{5, 14, 7},
{10, 9, 8}
};

8	[1][2]
9	[1][1]
10	[1][0]
7	[0][2]
14	[0][1]
5	[0][0]
a	

e.g. int[2][] = {
}

row size must be needed

Error

→ In, compile-time initialization, the array can be initialized in the source code.

Scanned by CamScanner

Note:
Here, the row-size must be defined.

2) Run-time Initialization:

e.g. int a[2][3];

→ In run-time initialization the array can be initialized at run-time.

→ In this process of initialization Both the row size and column size must be defined in the declaration part of the source code.

e.g.

int a[2][2];	error	int a[50][50];
a[3][3]		a[3][3]

2) WAP in 'C' to read the elements of a matrix and print it.

Ans.
```
void main( )
{
    int a[3][3], i, j;
    printf("Enter the elements of
            matrix:: \n");
    for (i=0; i<=2; i++)
    {
```

Scanned by CamScanner

```
for (j=0; j<=2; j++)
    {
        scanf ("%d", &a[i][j]);
    }
}
printf (" The matrix is :: \n");
for (i=0; i<=2; i++)
    {
        for (j=0; j<=2; j++)
            {
                printf ("%5d", a[i][j]);
            }
        printf ("\n");
    }
```

3	[2][2]
6	[2][1]
11	[2][0]
4	[1][2]
9	[1][1]
8	[1][0]
7	[0][2]
10	[0][1]
5	[0][0]

Output:

```
Enter the elements of matrix::
5 10 7 8 9 4 11 6 3
The matrix is::
    5   10   7
    8    9   4
   11    6   3
```

Scanned by CamScanner

WAP in 'C' to input a matrix and print all its diagonal elements.

```
void main()
{
int a[3][3], i, j;
printf("Enter the elements of
        matrix::\n");
for (i=0; i<=2; i++)
  {
    for (j=0; j<=2; j++)
      {
        scanf("%d", &a[i][j]);
      }
  }
printf("The ~~diagonal~~ matrix is::\n");
printf("The diagonal elements are::\n");
for (i=0; i<=2; i++)
  {
    for (j=0; j<=2; j++)
      {
        if (i==j)
          printf("%5d", a[i][j]);
```

Scanned by CamScanner

```
else
    printf("   ");
    }
  printf("\n");
  }

}

}
```

~~Elements~~

Output:

```
Enter the elements of matrix::
5   10   7   8   9   4   11   6   3

The matrix is::
5   10   7
8   9   4
11  6   3

The diagonal elements are::
5   .   .
.   9   .
.   .   3
```

Scanned by CamScanner

Dt: 31/10/19

String in C

→ It can be divide as a single character or a group of characters enclosed within double codes.

e.g. char a[ ] = "Hello";

Initialization of strings

A string can be initialized in two ways

1) Compile time Initialization
2) Run-time Initialization

1) Compile time Initialization

syntax: datatype String_variable [size] = "String";

e.g. char a[20] = "Welcome";

or,

char a[ ] = "Welcome";

or

char a[ ] = {'w', 'e', 'l', 'c', 'o', 'm', 'e'};

→ In compile time initialization, the string can be initialized in the source code.

→ Here, the "size" of the must be greater than of the string must

Scanned by CamScanner

be one greater than the no. of character present in the string due to presence of a null character "\n" at the end of the string.

w	e	l	c	o	m	e	\n

size = 8

2) Run-time Initialization :-

A string can be initialized at run time by using several input functions.

scanf( ) -

syntax: scanf("control string", string_variable);

e.g. char a[20];
scanf("%s", a);

Note:

1) By using this function, it can read the string upto the white space or blank space character.

```
char a[20]; // a = welcome to BLS
scanf("%s", a);
printf("%s", a);
```
output = welcome

Scanned by CamScanner

gets ( ):

→ This function is used to read a string including white space.

syntax: ~~char a[20];~~ ~~syntax:~~

gets (string_variable);

e.g.

```
char a[20];   // a=welcome to BLS
gets (a);
printf ("%s", a);
```

Output: welcome to BLS

→ To print a string 'C' supports different function such as

printf ( ):

This function is used to display a string.

syntax: printf ("control string", string_variable);

e.g. printf ("%s", a);

puts ( ):

This function is also used to print a string.

Scanned by CamScanner

syntax: puts (string_variable);

e.g. char a[20];

gets (a); // welcome Balasore

puts (a); // output: welcome Balasore

Q) WAP to read a string in welcome and print it.

W
WE
WEL
WELC
WELCO
WELCOM
WELCOME

Ans.
```
void main( )
{
char a[ ] = "WELCOME";
int i,j;
for (i=0; i<=6; i++)
    {
    for (j=0; j<=i; j++)
        {
        printf ("%c", a[j]);
        }
    printf ("\n");
    }
}
```

E	a[6]
M	a[5]
O	a[4]
C	a[3]
L	a[2]
E	a[1]
W	a[0]

Scanned by CamScanner

Q) WAP to read a string in welcome and print it.

W
WE
WEL
WELC
WELCO
WELCOM
WELCOME
WELCOM
WELCO
WELC
WEL
WE
W

Ans void main( )
{
char a[ ]="WELCOME";
int i, j;
for (i=0; i<=6; i++)
{
for (j=0; j<=i; j++)
{
printf ("%c", a[j]);

Scanned by CamScanner

```
    }
    printf("\n");
  }
  for (i=5; i>=0; i--)
    {
      for (j=0; j<=i; j++)
        {
          printf("%c", a[j]);
        }
      printf("\n");
    }
  }
```

* String Handling Functions:

1) strlen( ):

→ This function returns the length of the string in terms of an integer value.

syntax: strlen (string_variable);

e.g.
```
void main( )
  {
    char a[ ] = "welcome";
    int i;
    i = strlen(a);
```

Scanned by CamScanner

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

2) Strcat( ):

→ This function will return a string which is concataration of two string i.e. this function will append the contents of second string into the contents of first string.

syntax: strcat (string_variable 1, string_variable 2);

e.g.

```
void main( )
{
char a[ ] = "Hello";
char b[ ] = "Hai";
char c[50];
c = strcat (a, b);
puts(c);
}
```

H	E	L	L	O	\

H	A	I	\

Output:

H	E	L	L	O	\	H	A	I	\

Scanned by CamScanner

Note:

→ Here, the size of the string "c" must be greater than the ~~size~~ of size of "a" and "b".

3) Strcpy ( ):

This function will return a string which is copying from second string to first string.

Syntax: strcpy (String_variable1, String_variable2)

e.g.

```
void main( )
{
char a[ ] = "Welcome";
char b[30];
strcpy(b, a);
puts (b);
}
```

O/p:

Welcome

4) Strcmp ( ):

→ This function ~~retu~~ will return an integer value i.e. it will return a value 0, if the

Scanned by CamScanner

two strings are equal. Otherwise, it will return the difference of ASCII value of the first miss matching character of both the string.

syntax:

Strcmp (String_variable1, String_variable2);

e.g.

```
void main ( )
{
    char a[ ] = "Hello";
    char b[ ] = "Hai";
    int i;
    i = strcmp (a, b);
    printf ("%d", i);
}
```

O/P:
4

Q) WAP to enter your name and print it in reverse manner.

Ans-

```
void main( )
{
    char name[30];
```

Scanned by CamScanner

```
int i, l;
printf("Enter your name::\n");
gets(name);
l = strlen(name);
printf("Reverse of your name is::\n");
for (i=l-1; i>=0; i--)
{
printf("%c", name[i]);
}
}
```

O/P:

```
Enter your name::
Toffan Kumar Mahalik
Reverse of your name is::
kilahaM ramuk naffoT
```

Q) AP in 'C' to read a sentence and count the no. of words.

```
void main ( )
{
char name[30];
int i, l, c=0;
printf("Enter a sentence::\n");
gets(name);
```

Scanned by CamScanner

```
l = strlen (name);
for (i=0; i<=l-1; i++)
  {
    if (name[i]=' ')
     c++;
  }
printf ("The no. of words are :: %d", ++c);
}
```

O/P

```
Enter a sentence ::
Welcome to FM University
The no. of words are ::
5
```

Q) WAP in 'C' to read a sentence and count the no. of times the character "a" appears.

Ans-

```
void main ( )
  {
  char name[30];
  int i, l, c=0;
  printf ("Enter a sentence ::\n");
```

Scanned by CamScanner

```
gets (name);
L = strlen(name);
~~for (i=0; i<L~~
for (i=0; i<=L-1; i++)
  {
    if ((name[i]='a') || (name[i]='A'))
      c++;
  }
printf("The no. of words are :%d", c);
}
```

Dt: 7/11/19

2) WAP to show sum of two nos. with using fun^n and without using function.

```
void main ( )
  {
  int a, b, sum;
  printf("Enter two numbers:");
  scanf("%d%d", &a, &b);
  sum = a + b;
  printf("Sum is %d", sum);
  }
```

~~WAP to show sum of two nos. with using~~

```
int sum (int, int); // function declaration
void main( )
  {
```

```
int a, b, sum, d;
printf ("Enter two numbers:: \n");
scanf ("%d%d", &a, &b);
d = sum (a, b); // function calling
printf ("The result is %d", d);
}
int sum (int x, int y) // function definition
{
  int add;
  add = x + y;
  return add;
}
```

1.Q.) Symbols of flow charts with examples.

2.Q) Different conditional statements with their syntax and examples. if
if else

3.Q) Different looping statements with syntax and examples.

4.Q) Programs

1) Pallindrom
2) Greatest among three numbers.
3) Swapping of two number
4) Fibonacci number.

Q) Short Question

1) Rules for identifiers
2) Constants in 'C'

Scanned by CamScanner

3) Relational vs Logical Operators
4) Algorithm vs Flow chart
5) Precedence vs Associativity of operators
6) Data types in C
7) Assembly vs High level Language.

8) WAP in 'C' to find out greatest among two numbers using function.

Ans.
```
int great (int , int);
void main ( )
{
    int a, b, c;
    printf ("Enter two nos.:\n");
    scanf ("%d%d", &a, &b);
    c = great (a, b);
    printf ("The greatest is %d", c);
}
int great (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

Scanned by CamScanner

⊛ Function:

It is a process of dividing the whole program into small small modules.

→ Function is a self contest module or block of codes which performs a specific task.

→ In 'C', the function can be clarified into two categories i.e. Library fun, and user-defined functions.

The library functions are defined by the compiler and it is called inside a program.

e.g. printf( ), scanf( ), etc.

User-defined functions are defined by the user and it has 3 components:

1) Function declaration
2) Function Call
3) Function definition

⊛ Function declaration :-

→ This component of the function must be

Scanned by CamScanner

declared before it is invoked or before the function call.

→ This component is also known as function prototype.

→ In Function declaration, it has 3 parts:

a) Function type

b) Function name

c) List of Parameter types

In Function type, it indicates the type of the function.

In Function name, it indicates the name of the function.

In List of Parameter types, it indicates the type of parameters.

Syntax:

```
Type funⁿ_name (arg1, arg2, ...);
```

e.g. int sum (int, int) // sum of two integers.

2) Function Call:

→ This component can be used after funⁿ declaration.

→ In function call, it has two parts:

Scanned by CamScanner

a) Function name

b) List of actual parameters.

In function name, it indicates the name of the function.

A list of actual parameters, it contains the parameters.

Syntax: $\boxed{\text{fun_name (arg1, arg2, ...);}}$

E.g. sum(a, b);

Actual parameters

Function definition:

→ This component can be defined outside the main function.

→ This component has 3 parts:

1) Function type

2) Function name

3) List of formal parameters

In fun type, it indicates the type of the function.

In Fun name, it indicates the name of the function.

Scanned by CamScanner

A list of formal parameters, it contains the

Syntax:

```
Function_type  Function_name (arg1, arg2, ...);
```

E.g. int sum (int x, int y);

↳ Formal Parameters

Let us take an example to find out sum of two numbers using function.

1) WAP in C to find out the average of three numbers using function.

Ans.

```
int avg (int, int, int);
void main ( )
{
int a, b, c, d;
printf (" Enter three numbers:\n");
scanf ("%d %d %d", &a, &b, &c);
avg = (a+b+c)/3;  d=avg (a,b,c);
printf ("\n Average is %d", d);
}
int avg (int x, int y, int z)
{
int avg;
```

Scanned by CamScanner

```
    avg = (x+y+z)/3;
    return avg;
}
```

⊛ Categories of Function :-

The Function can be categorised into 5 types :-

1) Function with return type and with arguments
2) Function with return type and without arguments
3) Function without return type and with arguments
4) Function without return and without arguments
5) Function with multiple return types.

1) Function with return type and with arguments :-

→ In this category of function, it recieves the data from the calling function and similarly, the calling function recieves the data from the called function.

→ In this category of function, the function prototype can be declared as

Scanned by CamScanner

Syntax: $\boxed{\text{Type function_name (Type1, Type2, ...);}}$

e.g. int sum (int , int);

2) Function without return type and without arguments:

→ In this ~~type~~ category of function, it doesn't recieve any data values but returns a data value to the calling function.

Scanned by CamScanner

Syntax: $\boxed{\text{Type Function_name (void);}}$

e.g. int sum (void);

```
int sum(void);
void main( )
  {
    int result;
   result =sum ( );
   printf ("The sum is ::%d", result);
  }
int sum( )
   {
    int a, b, add;
  printf ("Enter two numbers ::\n");
  scanf ("%d%d", &a, &b);
  add = a+b;
  return add;
   }
```

3) Function without return type and with arguments:

→ In this category of function, it recieves the data values from the calling function but does not return any value to the calling function.

Syntax: $\boxed{\text{void Fun_name (Type1, Type2, ...);}}$

Scanned by CamScanner

e.g. void sum (int , int);

Program: Sum of two numbers

```
void sum (int , int);
void main ( )
{
   int a, b;
  printf ("Enter two numbers :: \n");
  scanf ("%d%d", &a, &b);
  sum (a, b);
}
void sum (int x, int y)
{
   int add;
   add = x+y;
   printf ("The result is :: %d", add);
}
```

4) Function without return type and without argument :-

→ Here, the function does not recieve any value from the calling function and similarly the calling function does not recieve any data value from called function.

e.g.

```
void display (void);
void main( )
{
   display ( );
   display ( );
```

Scanned by CamScanner

```
printf("Hello");
display();
display();
}
void display()
{
printf("* * * * * *");
}
```

Output:

```
* * * * * *
* * * * * *
Hello
```

5) Function return multiple value:-

→ A function can return multiple By the help of address operator (&) & indirection operator (*).

Recursion:

In this process a function is called itself.

e.g.

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

Output:

```
Hello
:
∞ times
```

Scanned by CamScanner

Q) WAP in 'C' to find out factorial of a no. using recursion.

With Recursion

```
int factorial (int);
void main( )
{
  int a;
printf ("Enter a number::\n");
scanf ("%d", &a);
printf ("Factorial is %d", factorial (a):");
  }
   int factorial (int x)
     {
       if (x==0)
       return 1;
     else
         return (
     }
```

Without Recursion

```
int factorial (int);
void main( )
{
  int a;
printf (" Enter a number:\n");
scanf (" Factorial is %d", factorial(a));
}
   int factorial (int x)
     {
       int fact = 1;
       while(x>0)
       {
         fact = fact * x;
       } x--;
```

Scanned by CamScanner

```
return fact;
}
```

⊛ Variables / Storage Classes:

→ Variable is also known as storage class and it can be classified into the following categories:

i) Local variable
ii) Global variable
iii) Register variable
iv) Auto variable
v) Static Variable
vi) Extern Variable

1) Local Variable:

→ If a variable is declared inside a function, then a variable is called "Local" for that function.

→ The scope of local variable is within the function.

e.g.
```
main()
{
    int a,b;
    - - -
    - - -
}
```

Here, a & b are local for "main()".

Scanned by CamScanner

2) **Global Variable:**

If a variable is declared outside i.e., Before the main( ), then the variable will be treated as "global".

The scope of global variable is flow out the program.

e.g.
```
int a=5;
main( )
{
    int b=6, sum;
    sum = a+b;
}
```

(*) **Note (Local variable):**

If a local variable is declared in the name of the global variable, then the global can be accessed by the help of scope resolution operator (::).

e.g.
```
int a=5;
main( )
{
    int a=10, b=6, sum;
    sum = ::a+b;
    printf("%d", sum);
}
```

Output: 11

3) **Register Variable:**

→ This variable can be declared by using a keyword "register".

Scanned by CamScanner

→ If an variable is declared as register then the value will be stored directly inside the register instated of memory location.

→ The register variable are faster than other variable.

e.g. register int a = 10;

4) Auto variable:

→ This variable can be declared by using a keyword "auto".

→ This variable are created or activated when the function is called, and they will be destroyed automatically after the execution of the function.

→ Automatic variables are also known as local variables.

```
e.g. main( )
     {
        int i, j, k;
       auto int p, q, r;
       ...
```

here, p, q, r are the automatic variable.

5) Static Variable:

→ The variable can be declared as static by using a keyword "static".

Scanned by CamScanner

If a variable is declared as static, then it is never be reinitialized in a single compilation time.

e.g. 1. static int x;

e.g. 2.
```
main ( )
{
    static int i=5;
    while (i>0)
    {
        printf ("Hello");
        i--;
        main ( );
    }
```

Output: Hello 5 times

6) Extern Variable:

→ Extern variable can be declared by using a keyword "extern".

→ Extern variable are also known as global variables.

→ Extern variable can be declared anywhere inside the program instead of before the main( ).

e.g.
```
main ( )
{
    int x,y;
    extern int a=10;
```

}

Here, 'a' is the extern or global variable.

UNIT-IV

POINTERS

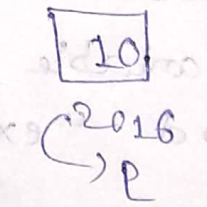
(*) Pointers:

→ It is a derived data types.

→ A variable is said to be as a pointer variable if it stores the address of another variable.

→ A pointer variable can be declared by using "*" symbol.

e.g. int a=10;
int *p;
p=&a;



→ The different advantages of pointer variable over general variable are as follows:

1) Pointers allows dynamic memory allocation/management.
2) Pointers reduce the length & time complexity of program.
3) Pointers increases the execution speed and thus, it reduces the program

Scanned by CamScanner

execution types.

// WAP in 'C' to find out sum of two number using pointers.

```
void main( )
{
    int a, b, sum;
    int *p, *q;
      p = &a;
      q = &b;
    printf("Enter two values::\n");
    scanf("%d%d", &a, &b);
      sum = *p + *q;
    printf("The result is::%d", sum);
}
```

// WAP in 'C' to swap two numbers using pointer.

```
void main( )
{
    int a, b, c;
    int *p, *q;
    p = &a;
    q = &b;
  printf("Enter two values::\n");
  scanf("%d%d", &a, &b);
        c = *p;
        *p = *q;
        *q = c;
```

Scanned by CamScanner

```
printf("After swapping a=%d, b=%d",
    *p, *q);
}
```

21/11/19

⊛ Rules for pointer operation:

→ A pointer variable can assign the address of another variable.

→ A pointer variable can be initialized with a null or zero value.

→ A pointer variable can be pre-fixed with increment and decrement operator.

→ Two pointer variable cannot be added.

→ A value cannot be assign to a pointer variable.

e.g. int *p;

p = 5; ✗

// WAP in 'C' to find out greatest among two numbers using pointers.

```
void main ( )
{
    int a, b;
    int *p, *q;
    p = &a;
    q = &b;
    printf("Enter two numbers :\n");
    scanf("%d%d", &a, &b);
```

Scanned by CamScanner

```
if (*p > *q)
printf("The greatest is :: %d", *p);
  else
printf("The greatest is :: %d", *q);
}
```

Call By Reference:

A function can be declared as pointer as its argument and this concept is called call by reference i.e., the functions are called by calling program with the address of a variable as arguments from it.

```
void swap (int *, int *);
void main()
{
  int a, b;
  printf("Enter two no. :: \n");
  scanf("%d %d", &a, &b);
  printf("Before swapping a = %d,
b = %d", a, b);
      swap(&a, &b);
  printf("After swapping a = %d, b = %d",
a, b);
```

Scanned by CamScanner

```
void swap (int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}
```

Call By Value	Call By Reference
→ In this method to call a fun^n only the value of the variable is passed as an argument.	→ In this method to call a function only the address of the variable is pass as an argument.
→ Memory location occupied by formal parameter and actual is different.	→ Memory location occupied by formal parameter & actual parameter is same.
→ There is no possibility of run data manupulation since the arguments are directly used in the express.	→ There is possibility of run data manupulation

e.g. void fact (int);
void main ( )
{
--- 	
fact(a);
}
void fact (int x)
{
--- 	
}
	e.g. void fact (int p)
	void main( )
	{
	fact(&a);
	}
	void fact (int *p)
	{
	--- 	
	}

Scanned by CamScanner

Dynamic Memory Management:

Dynamic Memory Management indicates the method of allocating a block of memory and releasing it whether memory is required at the time of running the program.

There are several function used for allocating the memory dynamically -

malloc( ), calloc( ), realloc( ), free( ).

1) malloc( ):

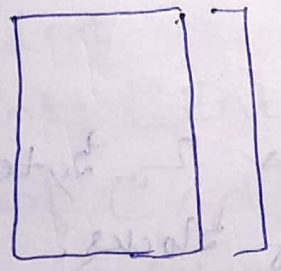
→ This function is used to allocate a single block of memory to store the values of specific datatypes.

Syntax:

```
int *ptr;
ptr = (datatype *) malloc (size);
```

e.g.

```
int *ptr;
ptr = (int *) malloc (80);
```



Memory

80×2
= 160 Bytes

Here, it will allocate 160 Bytes of memory space to the pointer variable "ptr".

Scanned by CamScanner

2) calloc ( ):

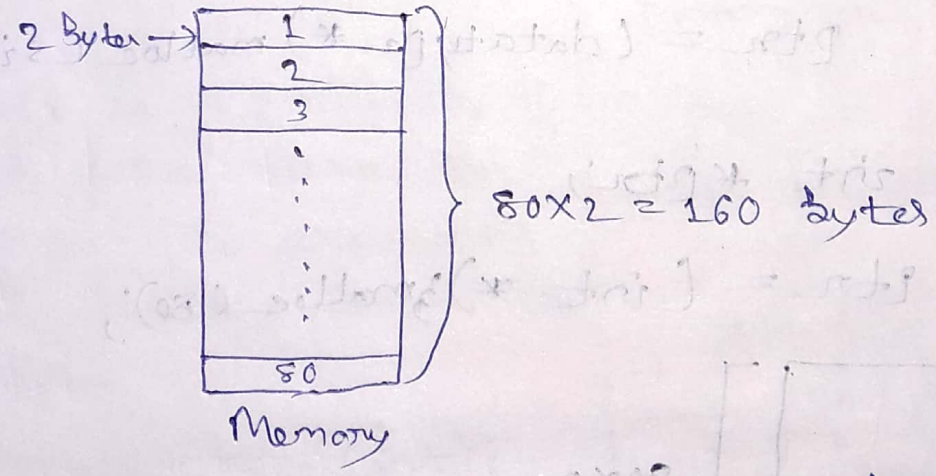
→ This function is used to allocate the memory in multiple blocks of same size during the program execution.

Syntax:
```
int *ptr;
ptr = (datatype *) calloc (n,m);
```

e.g. where, n = no. of blocks to be allowed

m = bytes allocated to each block

e.g.
```
int *ptr;
ptr = (int *) calloc (80,2);
```



Here, it is allocates 2 bytes of memory space to 80 no. of blocks.

3) realloc ( ):

→ This function is used to modify or reallocate, the memory spaces which is

Scanned by CamScanner

previously allocated to it.

Syntax:

```
ptr = realloc (old size, new size);
```

e.g.

```
ptr = realloc (80, 200);
```

4) free( );

→ This function is used to release the memory spaces which is allocated by using malloc(), calloc( ) and realloc( ).

Syntax:

```
free (pointer-variable);
```

e.g. free ptr;

Q) WAP in 'C' to read 'n' numbers from the keyboard and print them in reverse order by using pointer.

Ans.

```
void main ( )
{
  int
  printf("Enter the value of n:");
  scanf("%d", &n);
  ptr = (int *) malloc (100);
  printf("Enter the value:\n");
  for (i=0; i<n; i++)
  {
```

Scanned by CamScanner

```
        scanf ("%d", (ptr+i));
    }
printf ("The reverse elements are:\n");
for (i = n-1; i >= 0; i--)
    {
        printf ("%d", *(ptr+i));
    }
}
```

Output:

```
Enter the value of n:
5
Enter the values:
10 12 20 40 9
The reverse elements are:
9 40 20 12 10
```

* Structure:

→ It is derived data types or user-defined data type which consists of the data member related to a person or an item.

→ It is structure called abstract data type which can declared the mix datatype.

→ The main difference b/w array and structure is that array can declared homogenous data items, where as the structure is

Scanned by CamScanner

declared heterogenous data items.

→ A structure datatype is used to create the template of a particular item or a person.

→ Structure datatype can be declared by using a keyword "struct".

Syntax:

```
struct tagname
  {
    type datamember1;
    type datamember2;
    - - - - - -
    - - - - - -
  };
```

∴ where tagname indicates name of the structure.

type indicates datatype

datamember indicates the attributes.

e.g.

```
struct student
  {
    char name[30];
    int age;
    int roll;
  };
```

Scanned by CamScanner

Structure Variable:

→ The structure template can be asked through structure variable.

→ The structure variable can be declared either inside the structure definition or outside the structure definition.

~~Syntax:~~

e.g.1.

(inside the structure definition)

```
struct student
{
    char name[20];
    int age;
    int roll;
} stud1, stud2;
```

structure variable

e.g.2.

(outside the structure definition)

```
struct student
{
    char name[20];
    int age;
    int roll;
};
```

Scanned by CamScanner

~~void main( )~~

```
struct student
  {
    char name [20];
    int age;
    int roll;
  };
void main ( )
  {
    - - -
    - - -
  struct student stud1, stud2;
    - - -
    - -
  }
```

) WAP in 'C' to read name, age and roll no. of two students and print them using structure.

Ans.

```
struct student
  {
    char name [20];
    int age;
    int roll;
  } stud1, stud2;
void main ( )
  {
```

Scanned by CamScanner

```
printf("Enter the name, age, roll of 1st
        student::");
scanf("%s%d%d", stud1.name, &stud1.age,
        &stud1.roll);
printf("Enter the name, age, roll of 2nd
        student::\n");
scanf("%s%d%d", stud2.name, &stud2.age,
        &stud2.roll);
printf("The values of 1st student::\n");
printf("%s%d%d", stud1.name, stud1.age,
        stud1.roll);
printf("The values of 2nd student::\n");
printf("%s%d%d", stud2.name, stud2.age,
        stud2.roll);
}
```

* Initialization of structure variable:

Like other variable structure variables can be initialized in two ways:

i) Compile time Initialization
ii) Run time Initialization

i) Compile Time Initialization:

In this case, the structure variable can be initialize in the source code i.e., outside

Scanned by CamScanner

the structure definition.

e.g.
```
struct student
{
    char name[20];
    int age;
    int roll;
};
void main( )
{
    - - - -
    - - - -
    struct student stud1 = {"RAM", 19,
                            102};
```

Rules:-

1. We can not initialize the individual member within structure template.

e.g.
```
struct student
{
    char name[20];
    int age;
    int roll;
} stud1 = {"RAM", 19, 102};
```
Error

2. The order of value enclosed in braces must match with the members in the structure definition.

e.g.
```
struct student
{
    char name[20];
    int age;
    int roll;
```
Error

Scanned by CamScanner

```
};
- - - -
- - - -
struct student stud1 = {19, "RAM", 102};
```

Run Time Initialization:

→ In this process of initialization, the structure variable can be initialize at run time by giving the value from the Keyboard.

e.g1:
```
struct student
{
    char name[20];
    int age;
    int roll;
} stud1;
void main ( )
{
- - - -
- - - -
scanf("%s%d%d", stud1.name, & stud1.age,
        & stud1.roll);
- - - -
- - - -
```

Scanned by CamScanner

Array of Structure:

→ In this concept, we can define more than one structure variable using the concept of array.

e.g. Suppose we want to read the name, age & roll of ten students then we can use the concept of array of structure.

i.e.

e.g.

```
struct student
{
    char name[20];
    int age;
    int roll;
} stud[10];
```

Array within Structure:

→ This concept indicates an array is used inside a structure definition.

e.g. Suppose we want to declare the 5 subject marks of 10 students then we can use this concept.

```
struct student
{
    char name[20];
    int age;
    int marks[5];
} stud[10];
```

Scanned by CamScanner

⊛ Union :

→ It is also a programmer defined datatype like structure.

→ The main difference b/w structure and union is that in structure each data member is having its own memory location where as in union all the member use the same storage location that is maximum.

→ The union can be declare by using the keyword "union".

Syntax:

```
union tagname
  {
    type datamember 1;
    type datamember 2;
    - - - -
  };
```

ex:

```
union student
  {
    int age;
    int roll;
  } stud 1, stud 2;
```

↳ union variable

Scanned by CamScanner

File:

→ A file is a place on a disk where a group of related data item is stored, i.e., the file is a process of storing the data item on the disk.

→ In 'C', it supports a number of functions to perform the basic operation on file:

1) Naming a file
2) Reading a file
3) Reading the data from the file
4) Writing the data into file
5) Closing the file

'C' supports different library function to perform the operation on file such as:

1) fopen ( ):

This function is used to open a new file or to open a existing file for use.

Syntax:

```
FILE *f1;
f1 = fopen("file_name", "mode");
```

where, f1 = The file pointer

filename = A valid identifier name with

Scanned by CamScanner

or without extension

mode = The purpose of the file

FILE = The datatype to declare a file

e.g. f1 = fopen("FMU.txt", "r");

Different Modes:

Mode name	Explanation
r	open a file for reading
w	open a file for writing
a	open an existing file & to modify the data into the file.
r+	open a file for reading as well as writing
w+	open a file for reading as well as writing
a+	open a file for reading as well as writing

2) fclose( ):

→ This function is used to close a file which has been opened for use.

Syntax:

```
fclose (file_pointer)
```

e.g. fclose (f1)

Scanned by CamScanner

3) getc( ):

→ This function is used to read each character from the file.

4) putc( ):

→ This function is used to write each character into the file.

5) fprintf( ):

→ It will write a set of data value into a file.

6) fscanf( ):

→ This function is used to read a set of values from a file.

7) getw( ):

→ This function is used to read integer value from the file.

8) putw( ):

→ This function is used to write integer value into the file.

Q) WAP to create a file and print it contents on the monitor.

Ans- main( )

```
{
    FILE *f1;
    char c;
printf("Enter the characters into file:");
```

Scanned by CamScanner

```
f1 = fopen ("FMU", "w");
while ((c=getchar()) != '\n')
    putc (c, f1);
    fclose (f1);
  f1 = fopen ("FMU", "r");
printf ("The contents of the file are : \n");
  while ((c=getc(f1)) != eof)
      putchar();
    fclose (f1);
}
```

Scanned by CamScanner

Q) WAP in 'C' to create a new file, write some contents into the file and copy them and paste into another file.

Ans-
```
void main( )
{
    FILE *f1, *f2;
    char c;
    f1 = fopen ("FMU", "w");
printf ("Enter the character into the file:");
    while ((c=getchar( ))!='\n')
        putc (c, f1);
    fclose (f1);
    f2 = fopen ("BLS", "w");
    f1 = fopen ("FMU", "r");
while ((c=getc (f1)) != eof)
        putc (c, f2);
    fclose (f1);
    fclose (f2);
f2 = open ("BLS", "r");
while ((c=getc (f2)) != eof)
        putchar (c);
    fclose (f2);
}
```

Scanned by CamScanner

Structure	Union
→ It is a programmer defined data type.	→ It is also a programmer defined data type.
→ The structure can be defined by using a keyword "struct".	→ The union can be defined by using the keyword "union".
→ In structure, each member has its own storage location.	→ In union, all the members can share a common storage location i.e. maxima.
e.g. struct student { char name[20]; int age; int roll; } stud1, stud2;	e.g. union student { char name[20]; int age; int roll; } stud1, stud2;

* Macro:

→ Macro indicates the symbolic constants used inside the program.

→ Macro can be defined by the help of pre-processor directives "#define".

→ The pre-processor directives are executed before starting the compilation process.

Scanned by CamScanner

e.g.:-

```
# define FMU int;
void main( )
{
    FMU a = 10;
    printf("%d", a);
}
```

$\boxed{\text{Output} = 10}$

Scanned by CamScanner