C-Programming Notes

ICT 1st semester

by

Mukesh Singh

ICT 5th batch 2073

Sukuna Multiple Campus

Sundarharaincha-12, Morang

Mukesh Singh SMC

Unit-1

Introduction to Programming Concept

Page No.
Date

1.1 Introduction of Programming Language:

A computer cannot perform any work by its own. It must be provided with some instructions, by which it knows what is to be done. Collection of such instructions is called a program. A computer understands only limited set of instructions which must be in binary codes.

The communication between two parties whether they are machine or human beings, always need a common language or terminology. But the computer cannot understand human language like Nepali, English. It needs machine language. ~~So, such artificial languages~~

So, such artificial languages that are used to write a set of instructions (programs) for the computer are known as computer programming languages.

Some popular computer programming languages are C, C++, C#, JAVA, Pascal, etc.

$\boxed{HL \rightarrow PL \rightarrow LP \rightarrow ML}$

Computer Programming languages can be classified into two categories. They are:-

(i) Low level language (0,1) → Machine & Assembly
(ii) High level language

Programming Language

- Low Level Language (LLL)
  - Machine Language (0,1)
  - Assembly Language
- High Level Language (HLL)
  - Third Generation Language (3GL)
  - Fourth Generation Language (4GL)
  - Fifth Generation Language (5GL)

1.2. Assembler Compiler & Interpreter:

i. # Assembler:

Assembler is a language processor that translates assembly language program into machine language program. As computer understands only machine language, assembly language program must be translated into machine language program before execution. Otherwise, computer does not understand the codes.

Source Program (Assembly Program) → Assembler → Object Code OR (Machine Code)

ii. # Compiler:

A compiler is a language translator which translates high level language program into machine language program. Compilers are more efficient & faster

Mukesh Singh SMC

than interpreter. For each high level language, a separate compiler is required. For example, a C language compiler cannot translate a program written in JAVA programming languages. Programming languages such as C, C++, C#, Visual Basic etc. use compiler. JAVA has both compiler & interpreter.

High level Program (Source Code/Program)	→	Compiler	→	Object Code OR Machine Code

iii. # Interpreter:

An interpreter is a language translator which translates high level language program into machine language program one instruction at a time. Unlike compilers, it translates and executes the program line by line. Each time, the program is executed, every line is checked for syntax error and then converted to the equivalent machine code. Programming language, such as Q-BASIC, LISP, PERL etc. use interpreter.

Difference between Compiler & Interpreter:

Compiler	Interpreter
1. It translates high level language into machine language program at single attempt.	1. It translates high level language into machine language program by one instruction at a time.
2. It finds the syntax errors after compiling the whole program.	2. It finds the syntax errors after translating a line of the program at a time.
3. It is difficult to trace errors and causes of the errors.	3. It is easy to trace errors & causes of the errors.
4. The compiling process is faster than interpreter.	4. The interpreting process is slower than compiler.
5. It is more efficient than interpreter.	5. It is less efficient than compiler.
6. It creates the object code.	6. It ~~creates~~ does not create the object code.
7. Examples: C, C++, Visual Basic,	7. Examples: BASIC, LISP etc.

1.3. Syntax and Semantics:

1. # Syntax:

Syntax of a programming language defines the grammatical rules to construct an instruction in a program. Moreover, it also defines the alphabets, numbers, operators, expressions and structure of an instruction. This is the fundamental rule that must

be followed by a programmer. Otherwise, the language does not understand the codes of a program. If it is syntactically correct then the source codes are translated into machine codes.

ii# Semantics:

A semantic of a programming language defines the logical meaning of an instruction. Compiler does not identify the semantic of a program. So, it is very difficult to identify the semantic of a program. To identify the semantics of a program, a programmer might take some sample data to verify the outputs. We can consider the following English statements to know the idea of syntax and semantic.

Examples	Description
I eat rice	Syntactically and Semantically correct.
I eat house	Syntactically correct but Semantically incorrect.
I eats house	Syntactically & Semantically incorrect.

1.4. Program Design Tools:

During designing a program, a programmer designer requires different tools for solving a problem. Some of the most commonly used tools are algorithm, flowchart, pseudocode, etc.

1.4.1 Algorithm:

Algorithm is one of the most basic tools used for solving problem. It is defined as finite sequence of instructions for solving a problem. It consists of stepwise list of English statements making sequential procedure. The number of instructions should be minimized to increase the speed of algorithm. It is named after the 9th century Mathmatician, "Abu Jafar Muhammad Ibn Al-Khwarizmi of Baghdad (Iraq).

Rules for writing (developing) an algorithm:

- The number of instructions should be finite.
- The steps written in algorithm should be in simple language and unambiguous.
- Algorithm should be prepared by dividing the problem into small ~~scheduler~~ modules.
- It should have an input, process and desire output after executing the algorithm.

Advantages:

- It is very simple and easy to understand.
- It has no hard and fast rules for writing an algorithm.

Disadvantages:

- It is more difficult for translating algorithm into program codes.
- It is not considered as standard tool for program designing.

Extra knowledge

1.4.2. Flowchart:

It is defined as ~~the~~ a pictorial representation of an algorithm that describes the procedure to solve a problem. It shows the order of instructions and relationship between them. The purpose of constructing flowchart is to help for the programmer for understanding the logic of a program.

The first "flowchart" was developed ~~by~~ by Frank Gilberth.

A flowchart is classified into two types:

i. System flowchart
ii. Program flowchart

Advantages of flowchart:

- It is very efficient tool for solving logical & mathematical problems.
- It is easy for converting flowchart into program codes.
- It is easy for testing and maintenance.

Disadvantages of flowchart:

- It is difficult for solving large and complex problem.
- It is time consuming for designing a flowchart.
- It is very difficult for modifying and updating the flowchart.

Extra knowledge

Some Universal notations / symbols of flowchart:

* (Start / End (stop)) → Terminal

→ Terminal indicates the beginning and end of the program. It is 'oval' shaped. Start or End is written

inside it.

* Input/Output box

→ This symbol indicates input or output function. This notation is used to input data and to display result. The shape of Input/output notation is parallelogram.

* Processing box

→ This symbol indicates the function (information processing) like calculations or movements. It is rectangle in shape.

* Decision

→ This symbol is used to make logical decision (True or False). It gives yes or no result. The shape of this symbol is diamond.

* Connector:

→ Connector is used to join different parts of the flow-chart.

Mukesh Singh SMC

* ← → ↑↓ : Flowlines

→ These lines are used to show the flow direction of the program. We can show the sequence of the program with the help of such flow-lines.

Extra

Program Flowchart	System Flowchart
1. It is used for designing small sized logical and mathematical problem.	1. It is used for designing a large sized real world problem.
2. It gives specific view of control flow of a program.	2. It gives the broad overview of the system.
3. It shows the instruction flow of a program.	3. It shows the data flow of a system.
4. Program flowchart is designed by designer.	4. System flowchart is designed by system designer.

1.4.3 Pseudo Codes:

Dictionary meaning of pseudo-code is 'fake' or 'false'. Pseudocode is a kind of algorithm in which the instructions are expressed in more English like structure and mathematical expression. The instructions of pseudocode are similar to the program instructions. It has no specific rules for writing instructions but the instructions are very closer to high level language instructions.

Advantages:

- Complex and lengthy problems can be easily solved

Extra

by using pseudocode.

- It is easy for converting Pseudo code into program codes.
- It is independent of any programming languages.

Disadvantages:

Extra

- Designer should have prior knowledge of programming while writing pseudocode.
- It is more complex than algorithm.

Pseudo Codes for calculating simple Interest

Start
Input Principle, Time, Rate
Calculate Simple Interest = (Principle * Time * Rate) / 100
Output Simple interest
Stop / End.

1.5. Features of good Programme:

(i) Integrity
(ii) Clarity
(iii) Simplicity
(iv) Efficiency
(v) Generality
(vi) Modularity
(vii) Security

Mukesh Singh SMC

Few programs Using Algorithm and flowchart:

1. Write an algorithm and draw a flowchart to find out largest number among ~~three~~ two numbers.

Algorithm

Step 1: Start

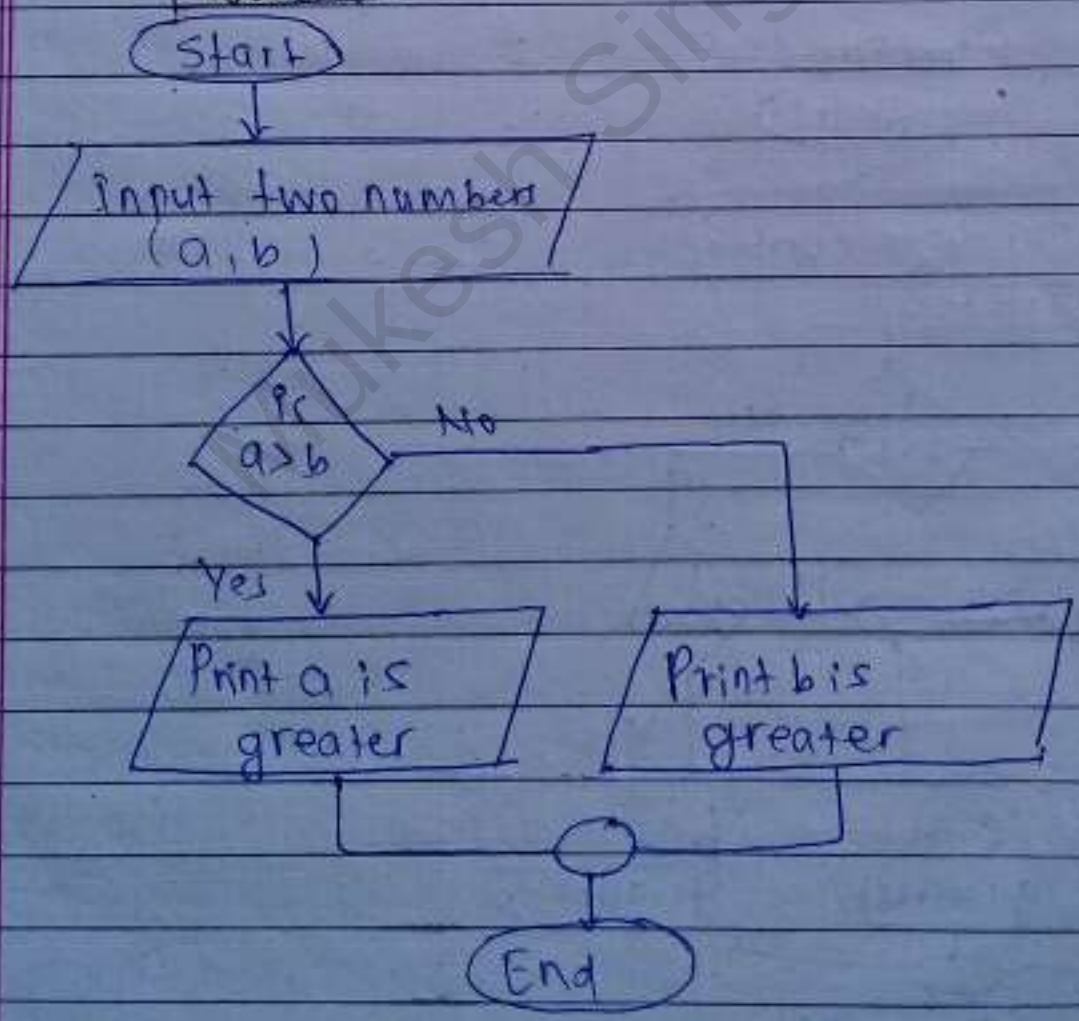
Step 2: Input two numbers a & b

Step 3: If a>b then a is greater
Else b is greater

Step 4: Print greater number

Step 5: End.

Flowchart:



Mukesh Singh SMC

Q. 2 Write an algorithm and draw a flowchart to find the largest number among three.

Algorithm

Step 1: Start

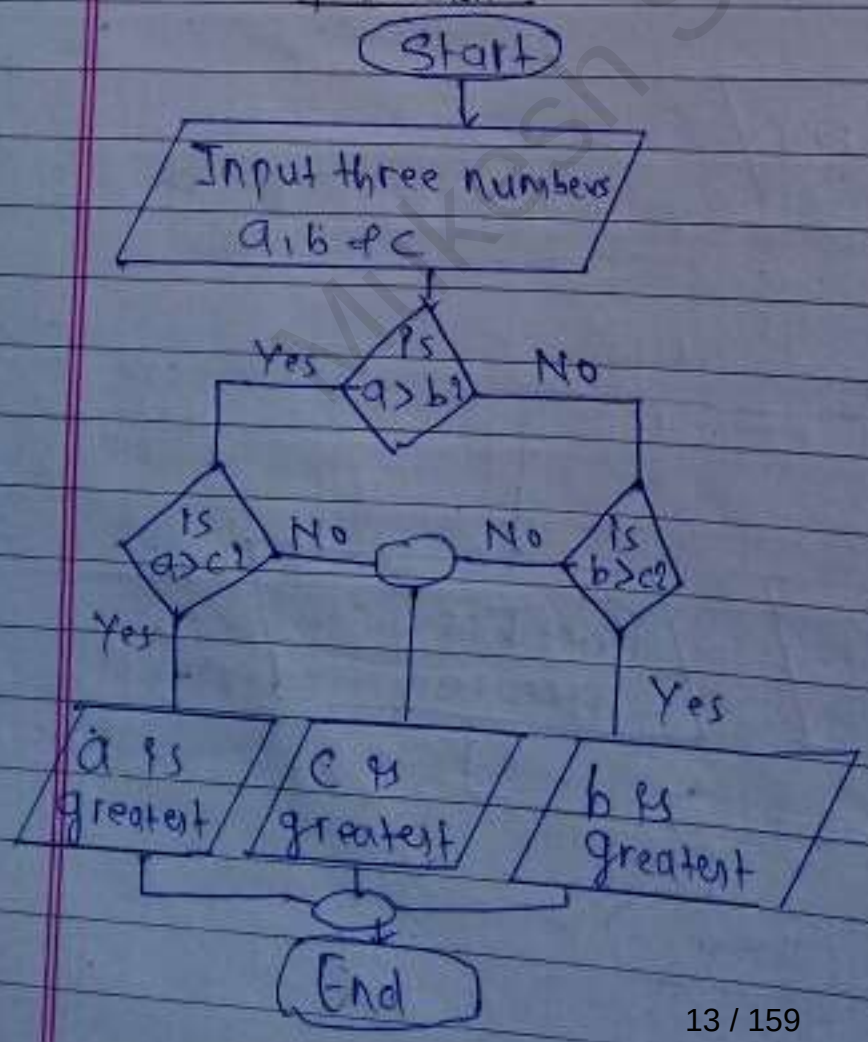
Step 2: Input three numbers a, b & c

Step 3: If a > b and a > c then
a is greater
If b > a and b > c then
b is greater
Else
c is greater

Step 4: Print greatest number

Step 5: End.

Flowchart



Mukesh Singh SMC

Q3. Write an algorithm and draw a flowchart to calculate area of a rectangle.

Flowchart

Algorithm

Step 1: Start

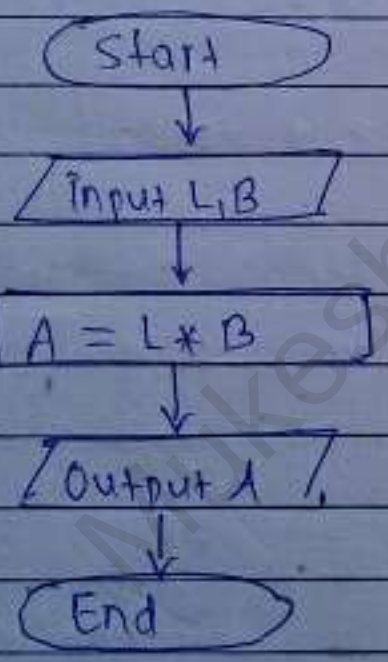
Step 2: Input L, B

Step 3: Calculate A = L * B

Step 4: Output A

Step 5: End

Flowchart



Mukesh Singh SMC

Q.4 Write an algorithm and draw a flowchart to calculate simple interest.

Algorithm:

Step 1: Start

Step 2: Input P, T, R

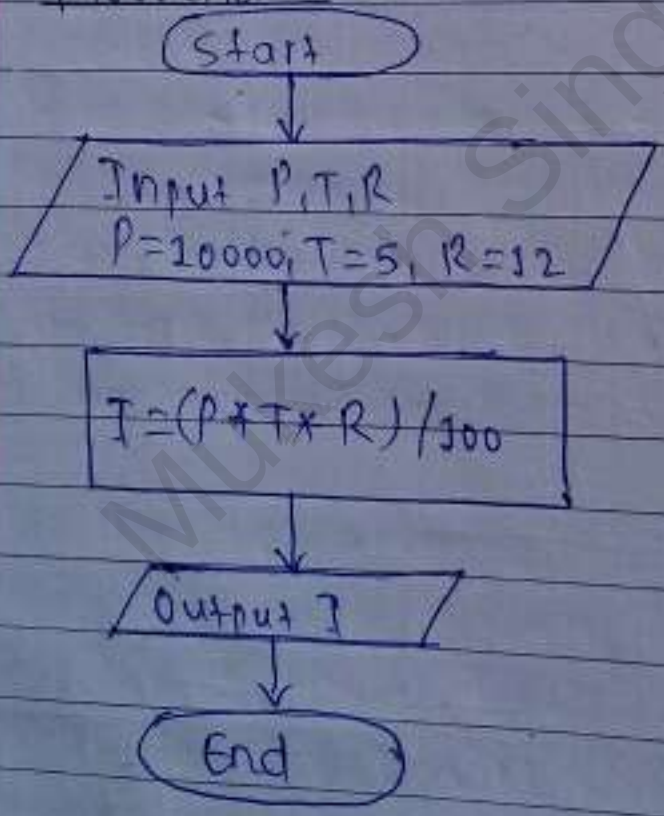
~~Step 3:~~ P = 10000, T = 5, R = 12

Step 3: Calculation I = (P * T * R) / 100

Step 4: Output I

Step 5: End

Flowchart:



Mukesh Singh SMC

Unit - 2

Introduction to C

2.1. History of C Programme:

C is a general purpose, portable, compact, simple, flexible programming language. C language was developed at AT and T's Bell Lab in 1972 by Dennis Ritchie. It is an offspring of the "Basic Combined Programming language" called BCPL, developed in the year 1967 at Massachusetts Institute of Technology. This language is associated with UNIX operating system. The source code for UNIX operating system is coded in C. It runs under a number of operating systems including MS-DOS. The C language is named as C because its precedence was called B which was derived from BCPL by Ken Thompson in 1970 in Bell labs.

Features of C Language:

C language is very popular language because it has large number of features listed below:

- It is highly portable language.
- It is structured programming language because the program is divided into number of functions.
- It is general purpose high level programming language.
- It has both features of high level as well as low level languages.

Advantages of C language:

- It is easy for debugging, testing and maintaining.
- It is portable programming language.

- It is fast for executing.
- It has only 32 keywords so that easy to remember.
- Its compiler is easily available.

Disadvantages of C- language:
- There is no runtime checking.
- As the program extends, it is very difficult to fix bugs(errors).
- It is difficult to understand because it is written by professional programmers.
- It does not support modern programming approach like object oriented programming.

2.2. Basic Structure of C- Programme:

Extra → Every C program contains a number of several building blocks known as functions. Each function performs task independently. The structure of C-program is given below:

* Header files:

C program depends upon some header files for function definition that are used in program. Each header file by default is extended with .h. The file should include using # include pre-processor directive.

for eg: # include <stdio.h>

* Global declaration:

It declares some variables that are used in

more than one function. These variables are known as global variables.

* void main()

Every program of C language must contain main() function. The main() is starting point of every C program.

* Declaration Part:

It declares variables that are used inside this particular function. The initializations of variables are also done in this section.

* Executable Part:

This part contains the statement blocks that may be input statement, process statement and output statement.

* User defined functions:

The functions defined by user are called user defined functions. These functions are defined before or after the main() function.

* Comments:

It is not necessary in the program. However, to understand the flow of programs, the programmer can include comments in a program.

Mukesh Singh SMC

* Function header:
It defines the heading or prototypes of the functions.

```
/* Comments */
Preprocessors and Header files
Function header
Global Declaration
void main()
{
Declaration Part
Executable Part
}
Sub-program section
function 1
function 2
------
------
function
```

User defined function

Mukesh Singh SMC

Example:

```
/* program to find sum of any two numbers */      → /* Comment */
#include <stdio.h>
#include <conio.h>                                  → Header files
int sum(int, int);                                  → Function header
int s;                                              → Global Declaration
void main()                                         → main()
{
int a, b;                                           → Declaration part
Printf("Enter any two numbers:");
Scanf("%d%d", &a, &b);
s = sum(a, b);                                      → Executable part
Printf("sum = %d", s);
getch();
}
int sum(int x, int y)
{                                                   → User defined function
int z
z = x+y
return(z);
}
```

Mukesh Singh SMC

Q. /* Write a program to display your name */

```
#include <stdio.h> // standard input output header file
#include <conio.h> // console input output header file
    void main()
    {
  Printf("My name is Mukesh"); // To display program
   getch();
   }
```

Q. /* Write a program to add two numbers (Numbers are a=10, b=15) */

```
#include <stdio.h>
#include <conio.h>
      void main()
      {
 int a, b, c;
      a = 10, b=15;
      c = a+b;
 Printf("Sum = %d", c);
   getch();
   }
```

Q. /* Write a program to find out simple interest (P=10000, T=5, R=12%) */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
int P,T,R,SI;
P=10000, T=5, R=12;
SI=(P*T*R)/100;
Printf("Simple Interest= %d",SI);
  getch();
}
```

OR By scanf method:

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
int P,T,R,SI;
Printf("Enter the value of P");
Scanf("%d", &P);
Printf("Enter the value of T");
Scanf("%d", &T);
Printf("Enter the value of R");
Scanf("%d", &R);
SI=(P*T*R)/100;
Printf("SI=%d", SI);
  getch();
}
```

OR

```
Printf("Enter value
    of P,T,R");
Scanf("%d %d %d", &P,
    &T, &R);
```

~~Output Enter value of P,T,R~~

Mukesh Singh SMC

Q. /* Write a program to read length and breadth of a room and calculate area and perimeter. */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
 int L,B,A,P;
Printf("Enter the value of Length");
Scanf("%d", &L);
Printf("Enter the value of breadth");
Scanf("%d", &B);
A = L*B;
P = 2*(L+B);
Printf(" Area = %d", A);
Printf ("Perimeter = %d", P);
  getch();
}
```

Q. /* Write a program to add, subtract, multiply & divide two numbers */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
int a,b, add, sub, mul, div;
a = 10, b = 5;
```

Mukesh Singh SMC

```
add = a+b;
sub = a-b;
mul = a*b;
div = a/b;
Printf(" add = %d", add);
Printf(" sub = %d ", sub);
Printf(" mul = %d", mul);
Printf(" div = %d", div);
  getch();
}
```

Q. /* WAP to read base and height of a triangle and print its area */

```
#include <stdio.h>
# include <conio.h>
    void main()
    {
int b, h, A;
 b=5, h=10;
 A= (b*h)/2;
Printf(" Area = %d", A);
 getch();
}
```

OR by scanf method:

```
# include <stdio.h>
# include <conio.h>
    void main
    {
 int b, h, A;
Printf("Enter base of triangle");
Scanf("%d", & b);
Printf("Enter height of triangle");
Scanf("%d", & h);
 A= (b*h)/2;
Printf(" Area = %d", A);
  getch();
}
```

Mukesh Singh SMC

Q. /* WAP to read radius of sphere and compute its surface area and volume*/

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
  int R, SA, V, Pi;
  Printf("Enter radius of sphere");
  Scanf("%d", &R);
  ~~Printf("Enter~~ Pi = 3.14;
   SA = 4*Pi*R^2;
   V = 4/3*Pi*R^3;
  Printf("Area = %d", SA);
  Printf("Volume = %d", V);
   getch();
    }.
```

Using float (%f).

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
    void main()
    {
  float R, SA, V;
  float Pi = 3.14;
  Printf("Enter value of radius) = ");
  Scanf("%f", &R);
  SA = 4*Pi*r*r; // 4*Pi*Pow(r,2);
  V = 4*Pi*r*r*r/3; //4*Pi*Pow(r,3)/3;
```

Mukesh Singh SMC

```
Printf("Area = %f", (A);
Printf("Volume = %f", v);
    getch();
}.
```

Q. /*WAP to input age of person and print in days with appropriate format*/

```
#include <stdio.h>
#include <conio.h>
  void main()
  {
 int Y, M, D, days;
Printf("Enter your year = ");
Scanf("%d", &Y);
Printf("Enter your month = ");
Scanf("%d", &M);
Printf("Enter your day = ");
Scanf("%d", &D);
  days = (Y * 365) + (M * 30) + D;
Printf("Total days = %d", days);
    getch();
}.
```

Mukesh Singh SMC

2.3. Character Sets, Token and Comments:

Character Set:

Character sets are valid sets of characters used in C-programming language. They include letters, numbers, and special symbols. The characters sets in C language can be classified into following categories:

- Letters/Alphabets: upper case A to Z (26) and lower case a to z (26)
- Digits : 0 to 9 = (10)
- Special Characters :

,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus Sign
?	Question Mark	+	Plus Sign
'	Apostrophe	<	Opening Angle (less than sign)
"	Quotation Marks	>	Closing Angle (Greater than sign)
!	Exclamation Mark	(	Left Parenthesis
|	Vertical bar	)	Right Parenthesis
/	Slash	[	Left Bracket
\	Backslash	]	Right Bracket
~	Tilde	{	Left Brace
_	Underscore	}	Right Brace
$	Dollar Sign	#	Hash
%	Percentage Sign		White space (34)

Total 128 symbols

Mukesh Singh SMC

Token:

The smallest part of C-programming language is called C tokens. C programs are written using these tokens. There are six types of ~~to~~ C tokens that are given below:

a. Keyword
b. Identifiers
c. Constants
d. Strings
e. Punctuators (Special Symbols)
f. Operator

a. Keywords:

Keywords are reserved words that are pre-defined in C programming language. Keywords cannot be used as identifiers in the program. Mainly there are 32 keywords used in standard C language. They are in following table below:

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

Mukesh Singh SMC

b. Identifiers:

Identifiers are the names given to program unit such as variable, structure, function, etc. They are not defined in programming language but are used to define by the programmer. Some basic rules to define identifiers are given below:

- First character must be alphabet or underscore than digits or alphabets.
- It must consist of only letters, digits and underscore.
- Any standard C language keyword cannot be used as identifier name.
- It should not contain a space.
- It allows both upper case and lower case character.

Valid identifier

A1 B34 FIRST_NAME X_1

Invalid identifiers

1A 344B int void FIRST_NAME X.1

Comments:

Comments are very useful components of program and they have no effect on program. They help programmer for the description of the program code. Comments can be written in two forms:

- // Single Line Comments
- /* Multi-line .....

..... Comments */

Mukesh Singh SMC

Page No.
Date

2.4. Variables and Constants:

Variables:

Variable are simply names/values that can be changed ~~any~~ the value while executing a program. Each and every variable must be declared before it is used. It is a memory location that can hold a value of certain data type. Programmers cannot use any of the keywords as variable names.

Eg: int a;

Here, int is data type and 'a' variable.

* Rules for naming variable:

- They must always begin with a letter (1-31 alphabets), digits or underscore.
- White spaces not allowed.
- A variable should not be a keyword.
- It should not contain any special characters.
- It does not allow keyword

There are two types of variables:

i. Numeric Variable

ii. String Variable

Constants:

Constant is a value (entity) that is never changed during the execution of a program.

Constants are also formed with alphabets,

Mukesh Singh SMC

numbers etc. However constant does not change but variables change. Consider following table with variable and constant in the memory cells:

Cell: | A | 5 | | A | 10 |

C supports several types of constants which are listed below:

i. Integer Constants
ii. Floating Constants (Real Constants)
iii. Character Constants
iv. String Constants
v. Symbolic Constants
vi. Escape Sequence Constants

2.5 Data Types in C

A data type in C programming language is a set of data with values (size) having predefined characteristics such as integers, floats, character etc. Storage representation and datatypes differ from machine to machine.

In C language, data types are classified into two categories:

(i) Primary data types
(ii) Secondary data types

i. Primary data types:

The fundamental data types are called primary data types. They are also used to build

Mukesh Singh SMC

other data types. The fundamental (primary) data types are given below in the following table:

Data Type	Size in bits	Size in Bytes	Range
Char	8	1	-128 to 127
Int	16-32	2-4	-32768 to 32767
float	32	4	
Double	64	8	
Void	0	0 (null)	

ii. Secondary data types:

Programmers can derive data types such as arrays, pointer, structure, union, enum etc. They are secondary data types.

2.6. Type Conversion

The process of converting one type of data into another type is called type conversion.

(i) Implicit Type Conversion

C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic type conversion is known as implicit type conversion.

Eg:

```
float x;
int x1 = 5;
int x2 = 2;
x = x1 / x2;
Printf("Output is %f", x);
```

Output is 2.000000.

ii. Explicit type Conversion:

Many times there may arise a situation where we want to force a type conversion in a way that is different from automatic conversion. The process of such a local conversion is known as explicit conversion or casting a value. The general form is (type-name) expression.

Example:

```
float x;
int x1 = 5;
int x2 = 2;
x = (float)x1/x2;
Printf("Output is %f", x);
```

Output is 2.500000

2.7 Input and Output

Like other languages, C does not have any built-in input/output statements as part of its syntax. All input/output ~~statements~~ operations are carried out through printf() and scanf() functions. There exist several functions that have more or less become standard for input and output operations in C. These functions (printf() & scanf()) are collectively known as standard (formatted) I/O functions because they can take any type of data.

C language consists of some I/O functions like

getchar(), putchar(), scanf(), printf(), gets(), puts(), getch(), putch(), getche() etc.

Extra

	Printf()	Scanf()
1.	It is used to display contents (text/number).	1. It is used to get inputs from the user.
2.	It contains any simple characters, format conversion specifiers or escape sequences and arg1, arg2 ---arguments (variables).	2 It does not contain simple characters, it contains formate conversion specifier and arg1, arg2, arg3 ..... arguments (variables).
3.	Syntax: printf("Control string", arg1, arg2 ....); Here control string may consist simple string or format conversion specifiers and arg1, arg2, ---- are arguments.	3 Syntax: Scanf("Control string", arg1, arg2...); Here control string may consist format conversion specifiers and arg1, arg2 ... are arguments.

4.	Example
	void main()
	{
	float a=987.55897;
	Printf("%.3f", a);
	}
		4 Example:
		void main()
		{
		int a;
		Printf("Enter a number");
		Scanf("%d", &a);
		}

Mukesh Singh SMC

2.8. Operators:

Operators are symbols that can be used for mathematical and logical operations such as +, *, <, - etc. Operators are used in C language to operate on data and variables. C has a rich set of operators which can be classified as:

- Arithmetic Operators (Binary operators)
- Logical Operators (Boolean operators)
- Relational Operators (Comparison operators)
- Increment / Decrement operators (Unary operators)
- Assignment Operators
- Bitwise Operators etc.

2.8.1 Arithmetic Operators:

All the basic arithmetic operations can be carried out in C. It can perform simple arithmetic calculations such as addition, subtraction, multiplication, division etc. (It uses two operands so that it's called binary operator). The types of arithmetic operators are given below:

Operators	Meaning	Examples
+	Addition	a+b
-	Subtraction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulus operator (Remainder)	a%b

2.8.2 Relational Operators:

It is required to compare the relationship between operands and bring out a decision accordingly. It is also called comparison operator because it is used to compare any two expressions. C supports the following relational operators:

Operators	Meaning	Examples
<	is less than	a<b
<=	is less than or equal to	a<=b
>	greater than	a>b
>=	greater than or equal to	a>=b
==	equal to	a==b
!=	not equal to	a!=b

2.8.3 Logical Operators:

Logical operators are used to give logical value either true or false. C has the following logical operators; they compare or evaluate logical and relational expressions:

Input		A && B	A || B	!A	!B
A	B	&&	OR	!	
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	0

Mukesh Singh SVC

OR

Operator	Meaning	Examples
&&	Logical AND	(a>b) && (a<c)
||	Logical OR	(a>b) || (a<c)
!	Logical NOT	!(a==b)

2.8.4 Increment / Decrement Operators (Unary operators):

The increment and decrement operators are very useful operators in C language. They are extensively used in loops. ~~The syntax of the operators is given below~~ Two unusual and unique operators ++ and -- called increment and decrement operators respectively. The use of these two operators will result in the value of the variable being incremented or decremented by unity 'i=i+1' can be written as 'i++' and 'j=j+1' can be written as 'j++' and 'j=j-1' can be written as 'j--'.

The increment increases the value of operand by 1; and the decrement operator decreases the value of operand by 1. They get only one operand, so called unary operators. The syntax for the operator is:

~~++a = a = a+1~~

	Example
1. ++variable name (increment prefix)	++a = a = a+1;
2. variable name ++ (increment postfix)	a++ = a = a+1;
3. --variable name (decrement prefix)	--a = a = a-1;
4. variable name -- (decrement postfix)	a-- = a = a-1;

The increment and decrement operators can be used as prefix or postfix notations. The operator can be placed either before or after the operands. If the operator is place before the variable (--i, ++i) known as prefix increment and vice-versa.

Examples

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
  int a=10;
Printf("a = %d\n", a); =10
Printf("a=%d\n", ++a); 11
Printf("a=%d\n", a++); 11
Printf("a=%d\n", a); =12
    getch();
    }
```

OR

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
  int a;
Printf("Enter the value");
Scanf("%d", &a);
Printf("a=%d\n", a);
Printf("a=%d\n", ++a);
```

Mukesh Singh SMC

```
Printf("a=%d\n", a++);
Printf("a=%d\n", a);
    getch();
}
```

9.8.5 Assignment Operators:

The assignment operator is used to assign the value of expression into variable. It is written in the form:

variable = expression

Eg: x = a+b

where 'x' is variable and 'a+b' is an expression. Here the value of a+b is evaluated and substituted to the variable x.

There are other forms of assignment operators which are listed in the table:

Operators	Statement with Operator	Equivalent to
+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

9.8.6 Bitwise Operators:

i. Bitwise logical operators
ii. Bitwise shift operators
iii. ~~Bitwise~~ One's Complement operator

Mukesh Singh SMC

i. Bitwise logical operators:

a. Bitwise AND (&)

b. Bitwise OR (|)

c. Bitwise exclusive OR (XOR |^)

a. Bitwise AND (&)

n1 = 16 = 10000

n2 = 10 01010

Bitwise n3 = 00000

b. Bitwise OR (|)

n1 = 16 = 10000

n2 = 10 = 01010

Bitwise n3 = 11010

~~Both input true = output false~~

c. Bitwise exclusive OR (XOR |^):

input		&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Both input true = output false

n1 = 60 111100

n2 = 15 001111

n3 = n1 ^ n2 (110011) ⇒ 110011

Mukesh Singh SMC

ii. Bitwise Shift Operators:

a. Left shift (<<)
b. Right Shift (>>)

a. Left Shift (<<)

The general form of left shift operators is
(operator << n)

n1 = 60

e.g. n1 = 00111100 (60)
Left shift 1 = 01111000 (120)
Left shift 2 = 11110000 (240)

b. Right Shift (>>)

The general form of right shift operator is
(operator >> n)

e.g. n1 = 60
n2 = n1 >> 60

n1 = 111100 = 60
Right shift 1 = 011110 = 30
Right shift 2 = 001111 = 15
Right shift 3 = 000111 = 7

Mukesh Singh SMC

Example of ~~shift~~ Bitwise operators:

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
 int n1=60, n2=15, AND, OR, XOR, leftshift, Rightshift;
  AND = n1 & n2;
  OR = n1 | n2;
  XOR = n1 ^ n2;
leftshift = n1 << n2;
Rightshift = n1 >> 2;
Printf("AND=%d", AND);
Printf("OR=%d", OR);
Printf("XOR=%d", XOR);
Printf("leftshift=%d", leftshift);
Printf("Rightshift=%d", Rightshift);
    getch();
  }
```

www.Sanjeev Singh SMC

Some C programs :

Q. /* ~~Prong~~ Program using arthmetic operator */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
int a=10, b=5, d, md;
    d=a/b;
    md=a%b;
Printf("d=%d", d);
Printf("md=%d", md);
 getch();
 }.
```

Q. /* WAP ~~to~~ ~~a~~ to use ~~Logical~~ relational operator ~~vs~~ (== & <) */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
int a=2, b=3, c, d;
 c=a==b;
  d=a<b;
Printf("c=%d", c);
Printf("d=%d", d);
  getch();
 }.
```

Mukesh Singh SMC

Page No.
Date

Using input (scanf)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c,d;
Printf("Enter the value of a");
Scanf("%d", &a);
Printf("Enter the value of b");
Scanf("%d", &b);
c = a==b;
d = a<b;
Printf("c=%d", c);
Printf("d=%d", d);
getch();
}
```

Q. /* WAP to use logical operators. The value should be taken from keyword */

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,x1,x2,x3;
Printf("Enter the value of a=");
Scanf("%d", &a);
```

Mukesh Singh SMC

```
Printf("Enter the value of b=");
Scanf("%d", &b);
   x1 = a<b && a>b;
   x2 = a<b || a>b;
   x3 = a<b ! a>b;
    getch();
   }.
```

Q. /* WAP using bitwise operator */

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
 int n1=16, n2=11, AND, OR;
   AND = n1 & n2; OR = n1 | n2;
   n1 = 10000;
   n2 = 1011;
   n3 = n1 & n2 ; n4 = n1 | n2;
 Printf("AND=%d", n3);
 Printf("OR=%d", n4);
    getch();
   }.
```

Mukesh Singh SMC

Q. /* WAP using bitwise operator */

```
#include<stdio.h>
#include<conio.h>
    void main()
    {
int n1=60, n2=16, AND, OR, XOR, leftshift, Rightshift;
   n1 = 111100;
   n2 = 10000;
AND = n1 & n2;
OR = n1 | n2;
XOR = n1 ^ n2;
leftshift = n1 << n2;
Rightshift = n1 >> n2;
Printf("AND = %d", AND);
Printf("OR = %d", OR);
Printf(" XOR = %d", XOR);
      getch();
    }
```

Mukesh Singh SMC

Unit-3

Control Structure

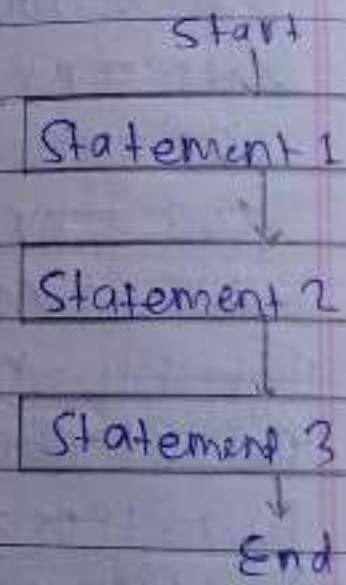
The structures which controls the flow of program from one part to another depending on condition are called control structure.

Types of Control Structure:

1. Sequential structure
2. Selective Structure (Branching Structure)
3. Looping Structure (Iteration Construct)

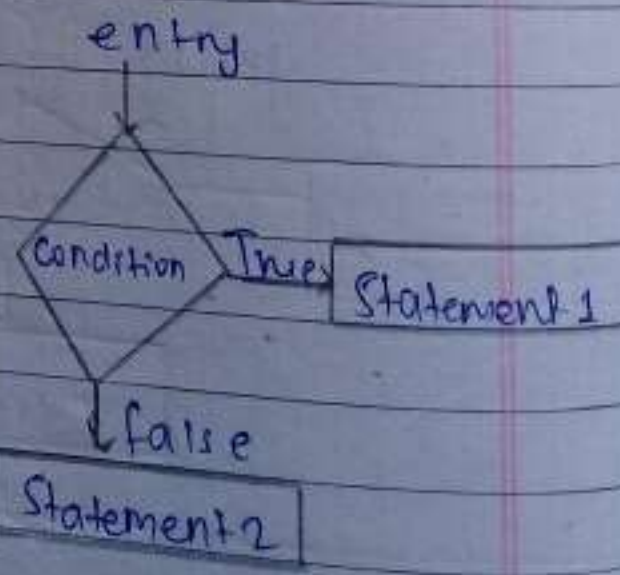
1. Sequential Structure

It is the most simplest programming structure where statements are executed sequentially from top to bottom without repetion, branching and without any condition.



(B.1) 2. Selective Structure:

In this structure, execution of statement depends upon a condition. If condition is true, one action is followed, otherwise another action is followed. It is also known as branching structure or decision structure.



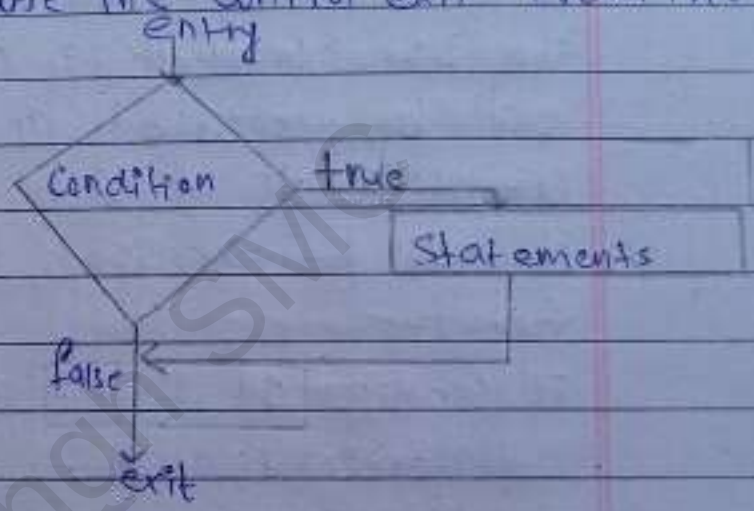
Mukesh Singh SMC

3.1.1. If statement:

If a given problem has one condition and only one action, then we can use simple if statement. We should consider either true part or false part. If the given condition is true then the statement will be executed otherwise the control exit from the condition.

Syntax:

```
if (condition)
  {
  statements;
  }
```

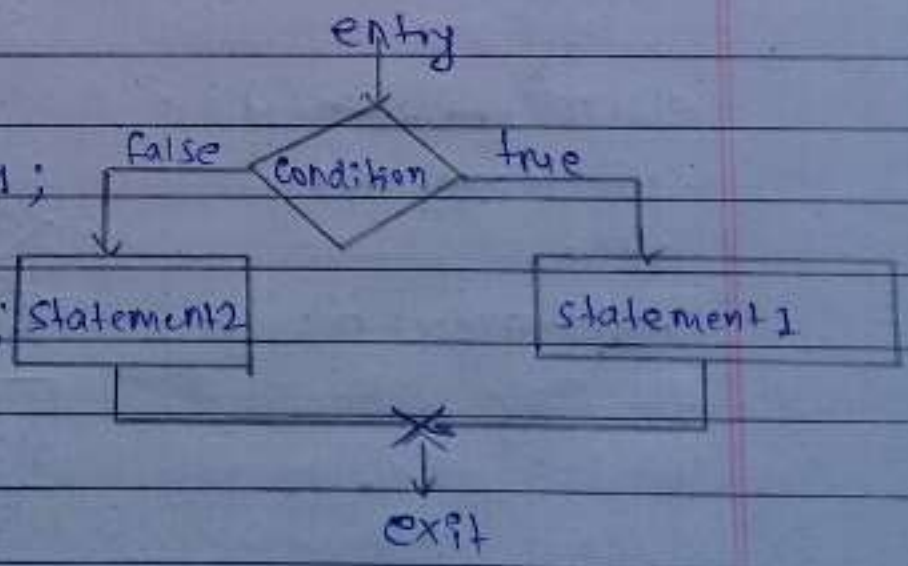


3.1.2 If-else statement:

Syntax:

```
if (condition)
  statement 1;
  else
  statement 2;
```

Flowchart



→ In this case, if the condition is true then statement 1 executed otherwise statement 2 is executed. The else portion of the if-else statement is optional.

Mukesh Singh SMC

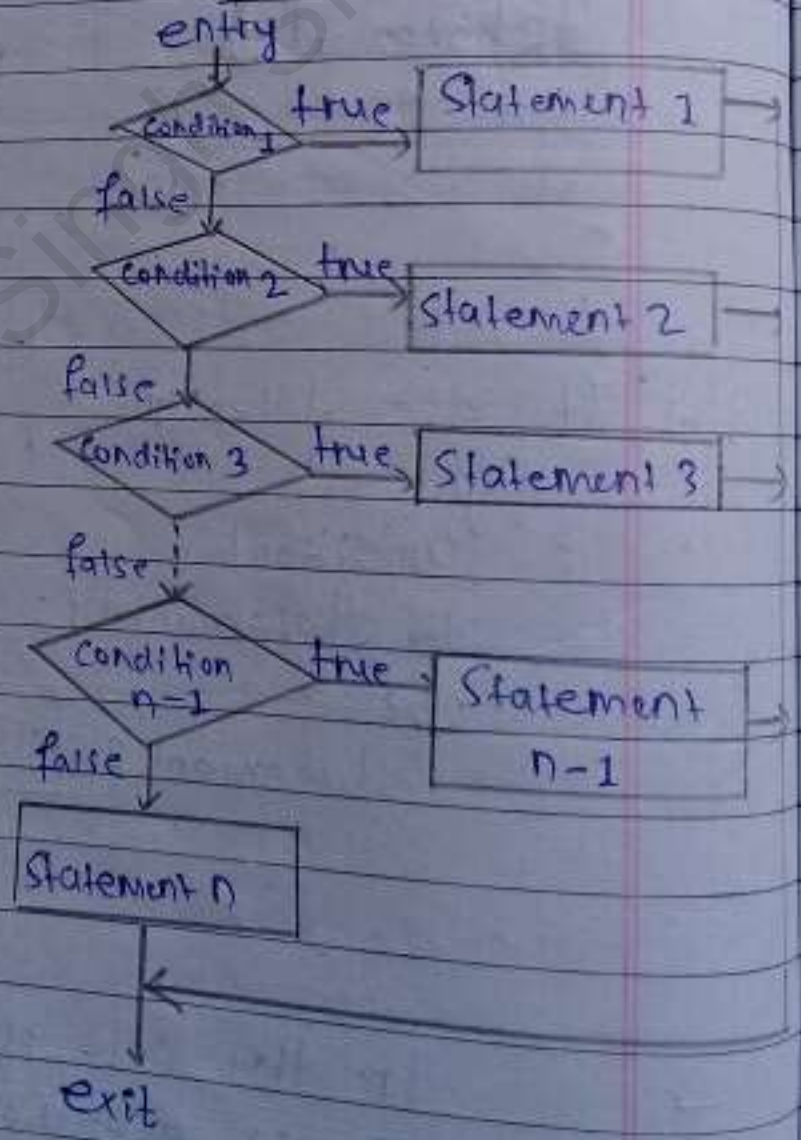
~~Nested~~ If-else if statement :

It is also known as ladder type if else. If a given problem has more than one interrelated conditions with respective actions then, we can use if-else statement. In this statement, if condition 1 is true, statement 1 will be executed otherwise condition 2 is checked, if condition 2 is true, statement 2 will be executed and similarly goes on for next conditions. If all conditions are false then last statement n will be executed.

Syntax:

```
if (condition 1)
  statement 1;
else if (condition 2)
  statement 2;
      :
      :
else if (condition n-1)
  statement n-1;
else
  statement n;
```

Flowchart



Mukesh Singh C

Page No.
Date

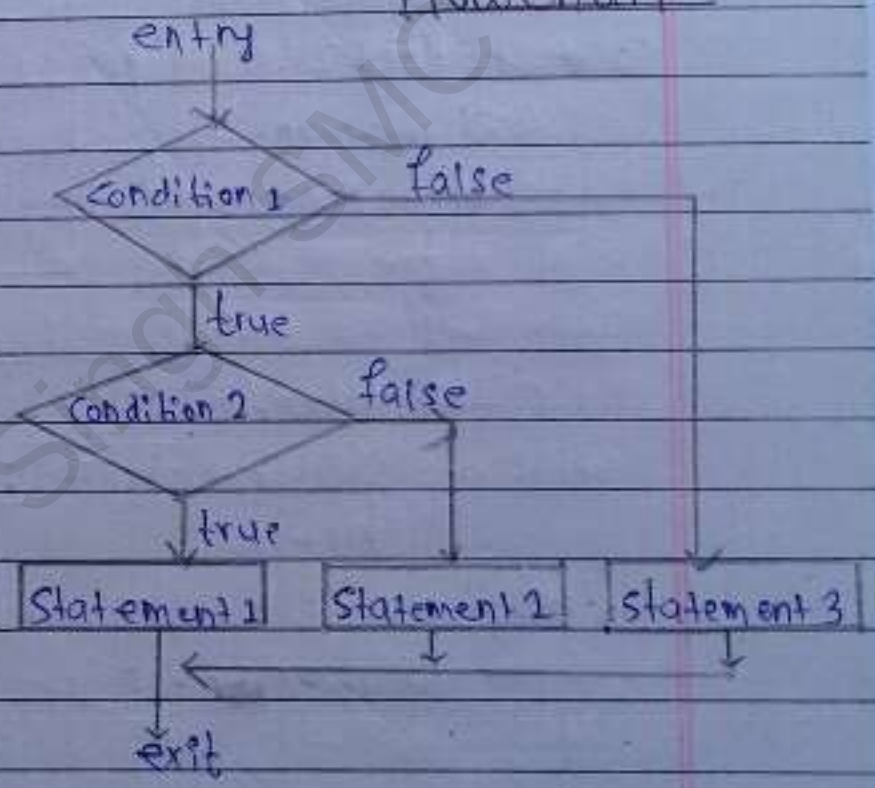
3.1.3. Nested if-else statement:

An entire if else statement written within the body if part or else part of another if else statement is called nested if else statement. It is used when a condition is to be checked inside another condition at a time in the same program to make decision.

Syntax

```
if(condition 1)
{
 if (condition 2)
   statement 1;
 else
   statement 2;
}
else
  statement 3;
```

Flowchart



3.1.4 Switch case Statement:

Switch is the multiple branching statement. If a given problem has one condition and respective more than two action then we can use switch case statement. It checks the value of the variable to the case value and then statements associated with it will be executed. If expression doesn't match any of the case value then the default

Mukesh Singh BMC

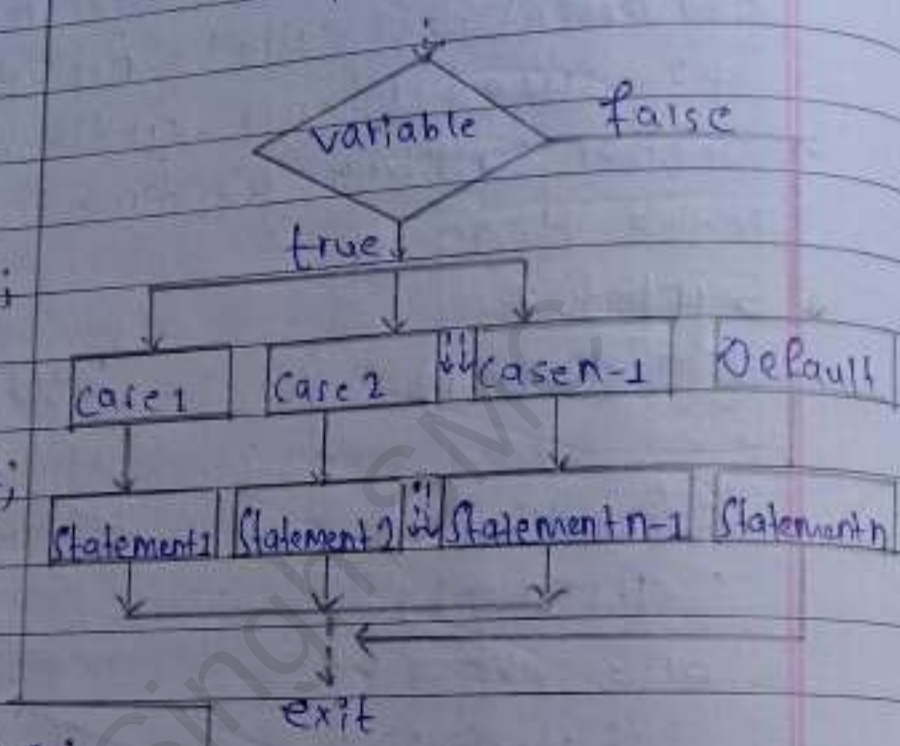
statement will be executed.

Syntax

Algorithm

```
switch(expression)
{
Case constant 1 :
        statement 1;
        break;
Case constant 2 :
        statement 2;
        break;
        :
        :
Case constant n-1 :
        statement n-1;
        break;
default :
        statement n
}
```

Flowchart ~~Flowchart~~



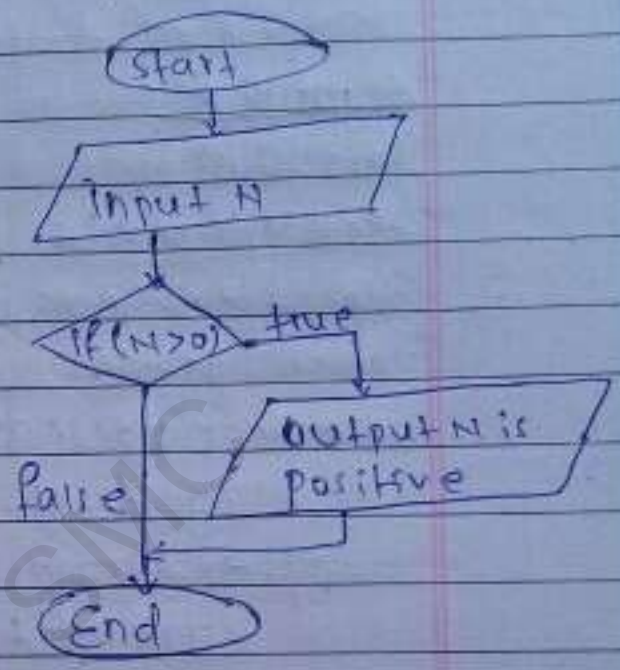
Mukesh Singh SMC

* Example of 'if' statement:

Algorithm

Step 1: Start
Step 2: Input N
Step 3: if (N>0)
    ~~Output~~ N is positive
Step 4: Output N
Step 5: End

Flowchart



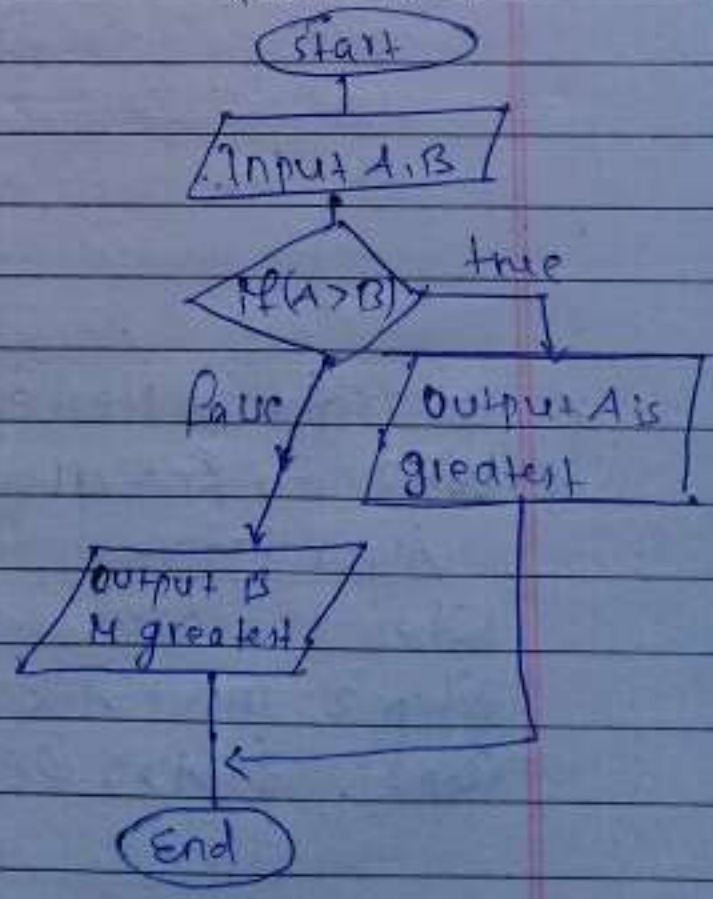
* Example of if else statement:

• WAP to find greatest number among two numbers:

Algorithm

Step 1: Start
Step 2: Input A, B
Step 3: if (A>B)
    ~~Output~~ A is greatest
    else
    B is greatest
Step 4: Output greatest number
Step 5: End

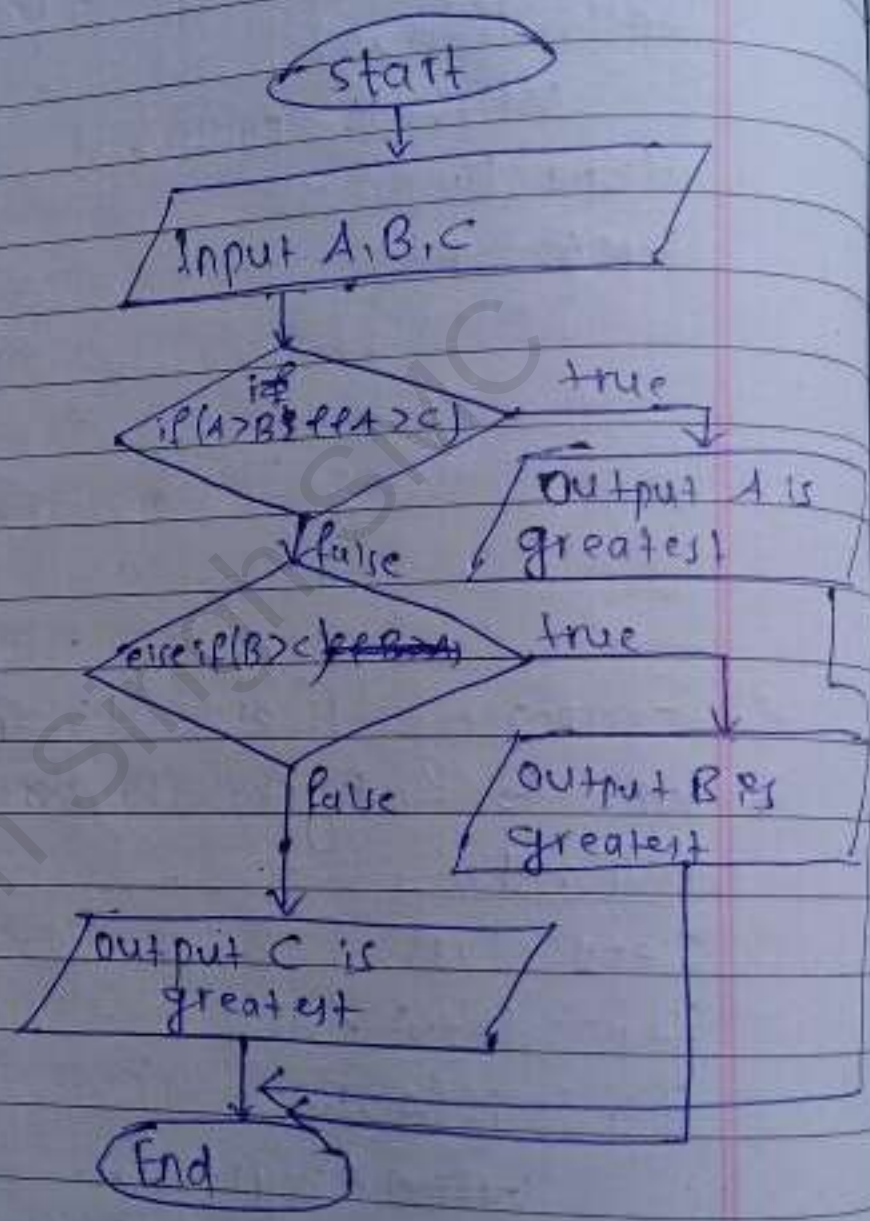
Flowchart



Mukesh Singh GMC

* Example of if-else if statement:
Find greatest number among three number:

Flowchart



Algorithm
Step 1: Start
Step 2: input A, B, C
Step 3: if (A>B && A>C)
A is greatest
else if (B>C && B>A)
B is greatest
else
C is greatest
Step 4: Print the greatest number
Step 5: End

* Example of Nested if else statement:
Q WAP to find greatest two positive numbers:
Algorithm
Step 1: Start
Step 2: Input A, B
Step 3: if (A>0 && B>0)
{

Mukesh Singh

~~Algori~~

if (A>B)

~~out~~ A is greatest

else

~~Output~~ B is greatest

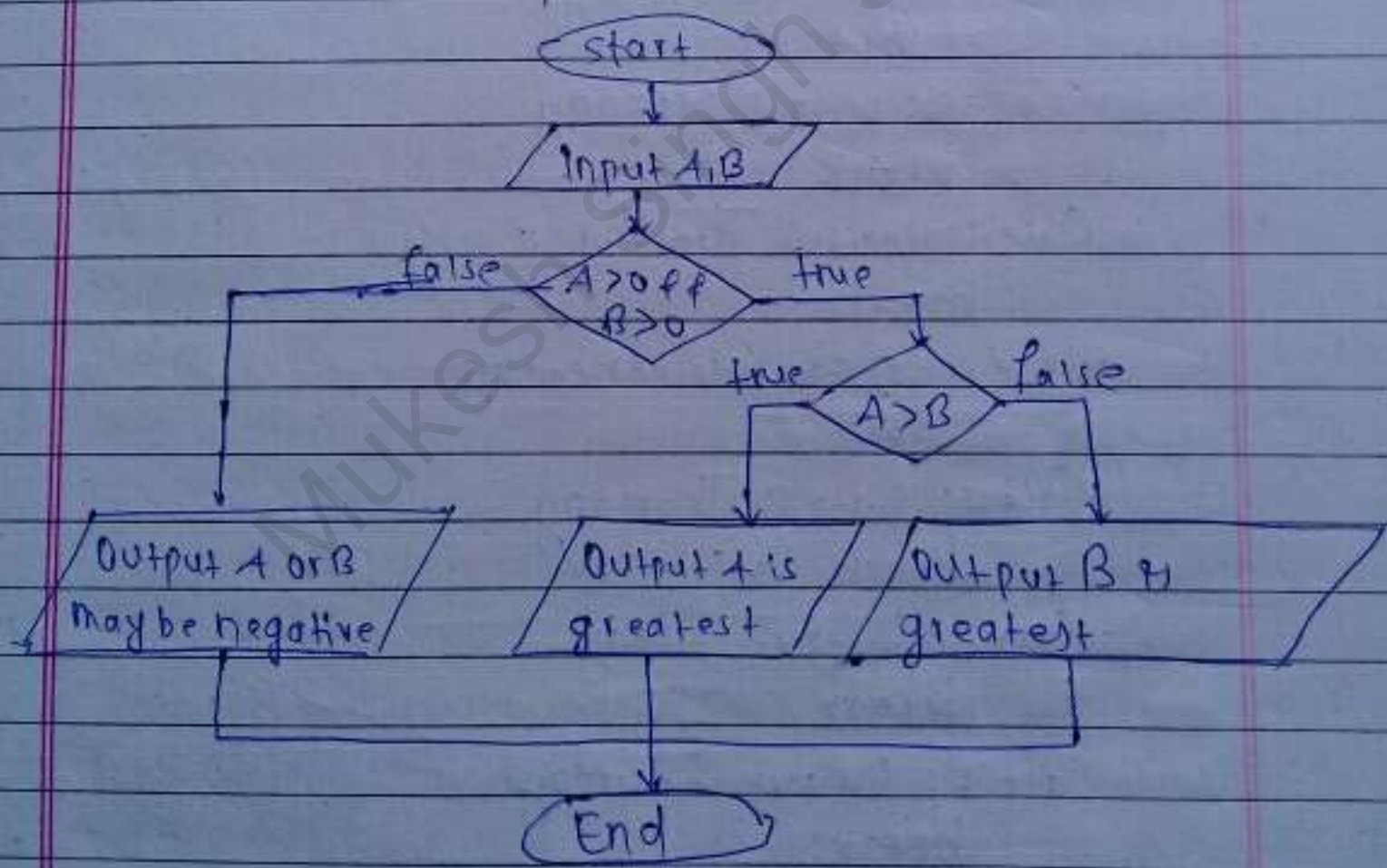
}

A or B may be negative

Step 4: Print two greatest positive numbers

Step 5: Stop

Flowchart



Mukesh Singh SMC

* Example of Switch case statement:

~~Algorithm~~ Write an algorithm and draw a flowchart that takes integer value 1 to 7 and print respective day: 1-Sunday, 2-Monday, etc.

Algorithm

Step 1: Start

Step 2: input day

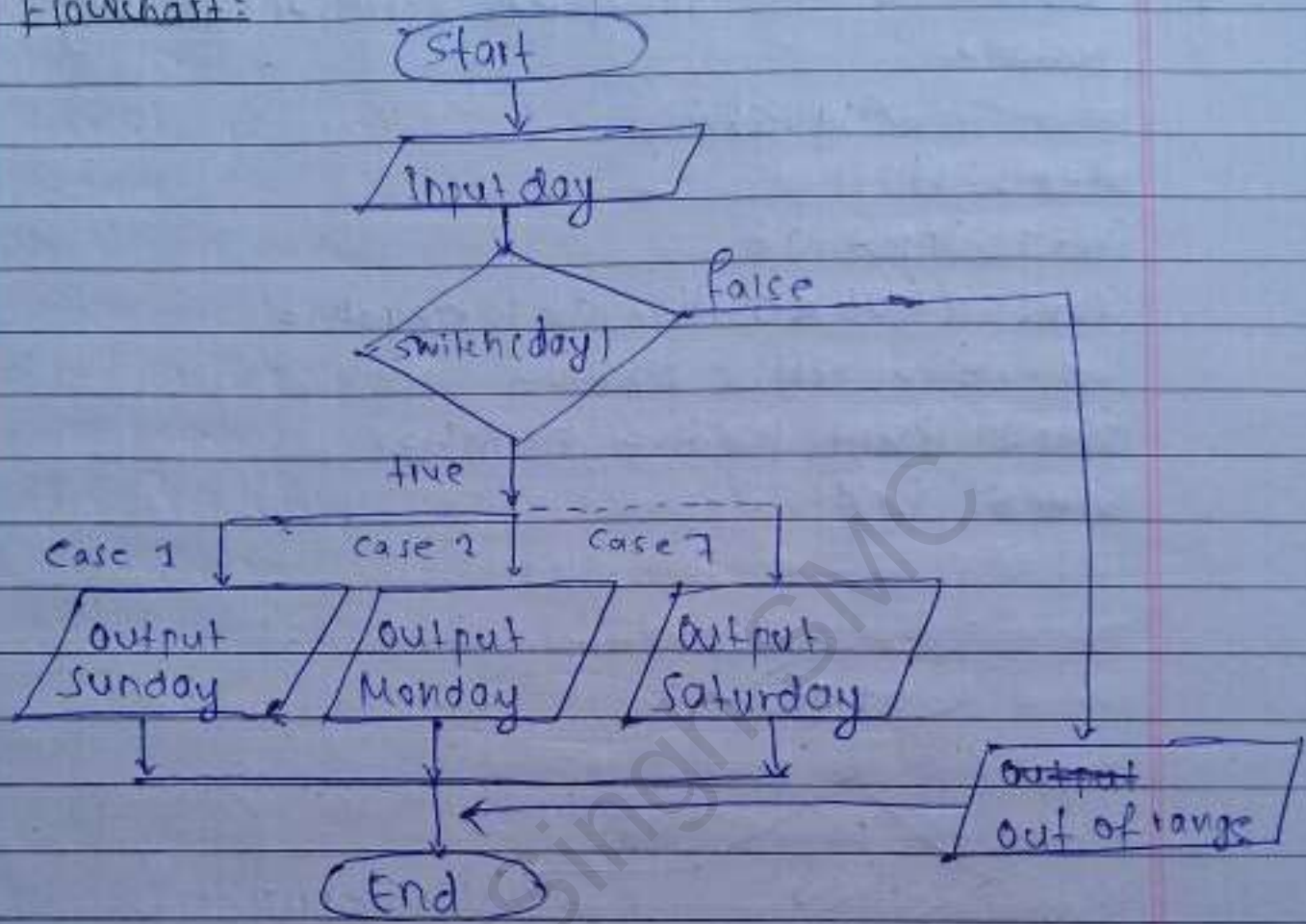
Step 3: Switch (day)

```
{
    Case 1: output Sunday
            break
    Case 2: output Monday
            break
    Case 3: output Tuesday
            break
    Case 4: Output Wednesday
            break
    Case 5: Output Thursday
            break
    Case 6: Output Friday
            break
    Case 7: Output Saturday
            break
    default: output out of range
}
```

Step 4: End

Mukesh Singh SMC

Flowchart:



Practical Questions:

1. Write algorithm, flowchart and C-program to find out the given number is negative or positive.
2. Write algorithm, flowchart and program to find out the given number is odd or even.
3. Write algorithm, flowchart and program to ~~find out~~ read three integer numbers and print the maximum.
4. Write a flowchart and a program to find out whether the given 4 digit number (year) is a leap year or not.

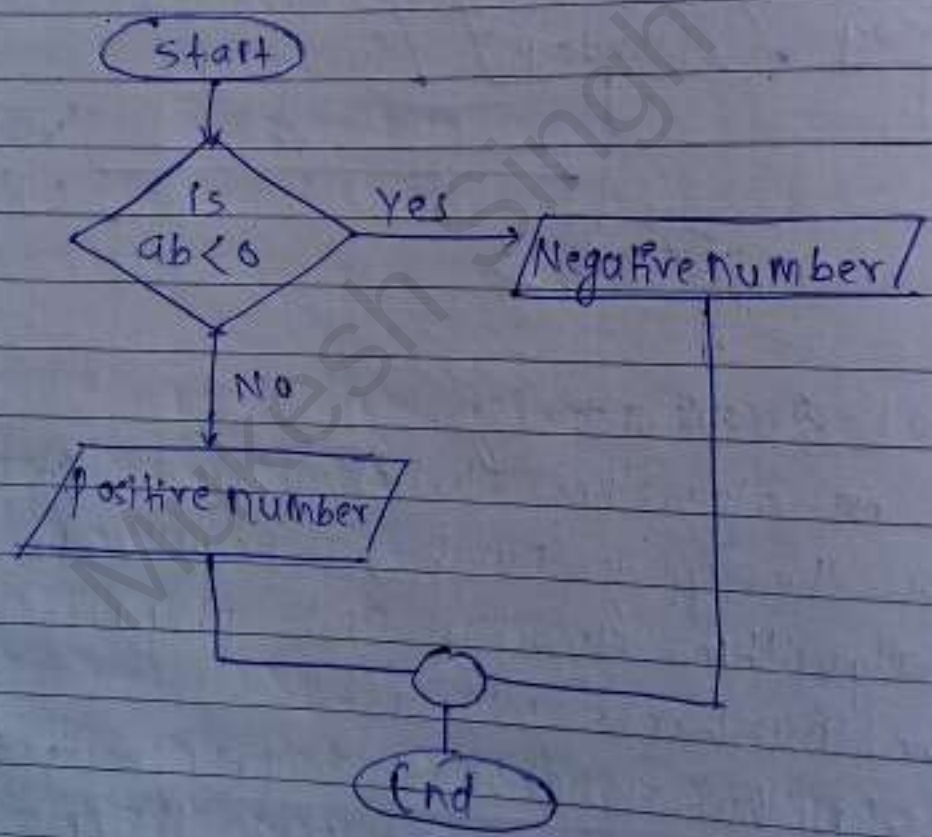
Mukesh Singh SMC

1 WAP to find out given number is negative or positive.

Algorithm

Step 1: Start
Step 2: Input ab
Step 3: If ab<0, then ab is negative
else ab is positive
Step 4: Print positive number
Step 5: End

Flowchart



C program

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

Mukesh Singh SMC

```
int ab;
printf("Enter a number = ");
scanf("%d", &ab);
if(ab<0)
printf("The number is negative");
else
printf("The number is positive");
getch();
}
```

9. WAP to find out given number is odd or even :

Algorithm

Step 1: Start

Step 2: Input A

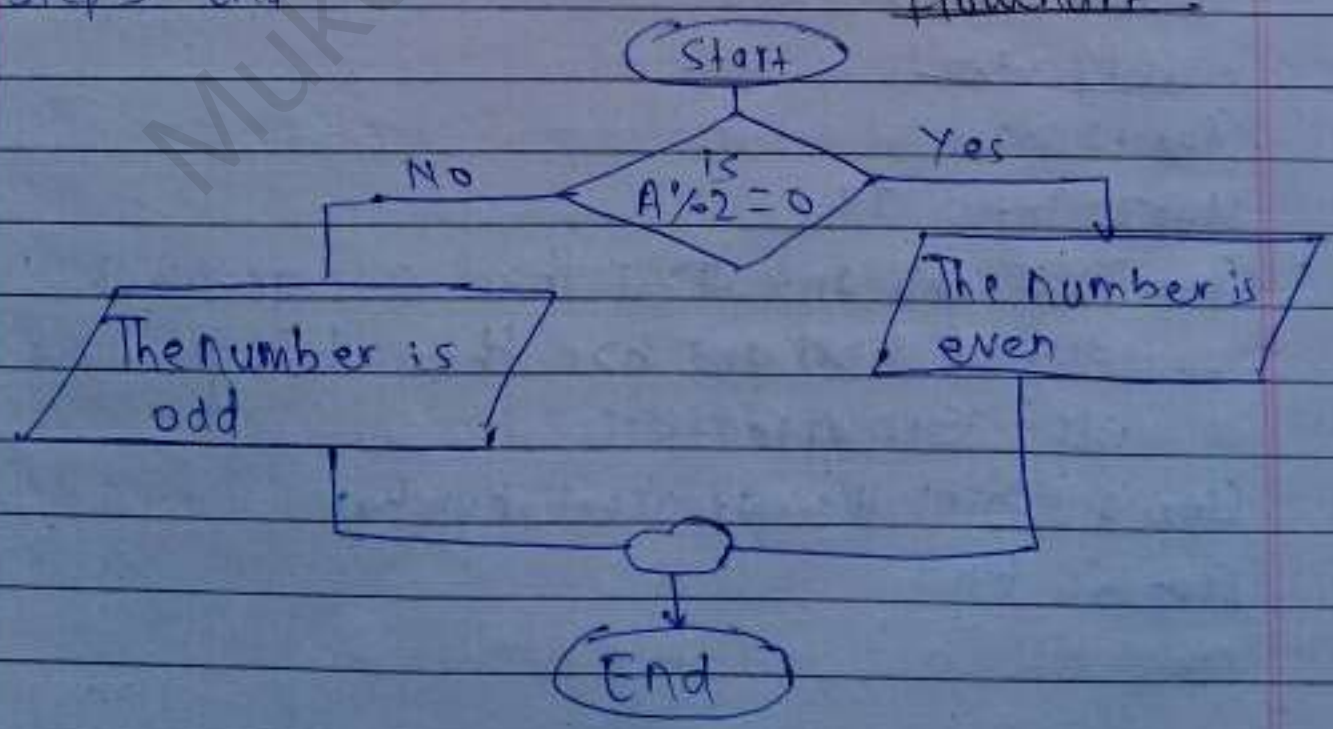
Step 3: if A%2=0 then ab is even

else A is odd

Step 4: Print odd number

Step 5: End

Flowchart



Mukesh Singh SMC

C-program

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

3. WAP to read three integer numbers and print the maximum.

Algorithm

Step 1: Start

Step 2: Input a,b,c

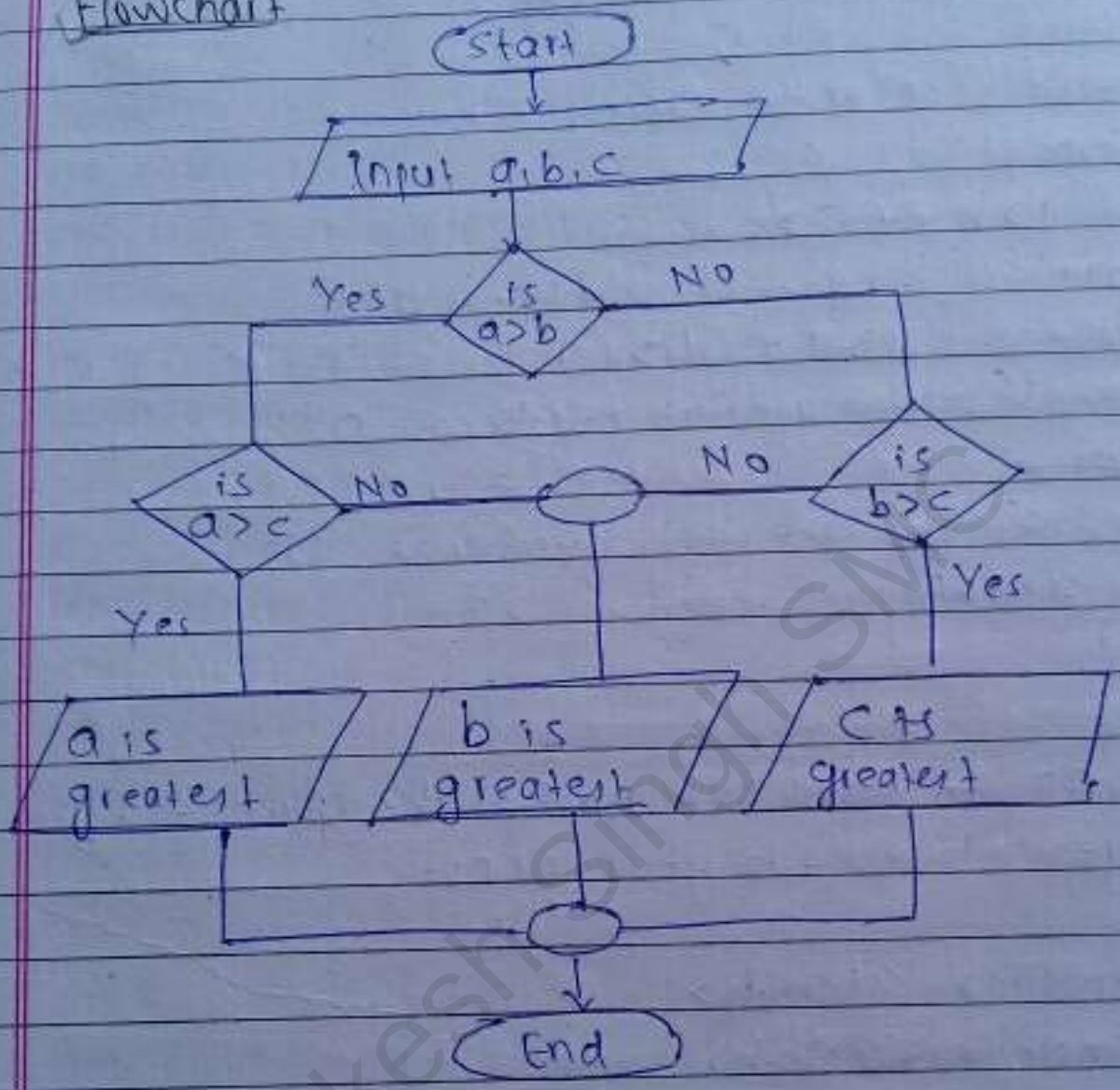
Step 3: if a>b and a>c then a is greatest
else if b>a and b>c then b is greatest
else c is greatest

Step 4: Print the greatest number

Step 5: End

Mukesh Singh SMC

Flowchart



C-Program

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
int a,b,c;
printf("enter the value of a");
scanf("%d", &a);
```

Mukesh Singh SMS

```
printf("enter the value of b");
scanf("%d", &b);
printf("enter the value of c");
scanf("%d", &c);
if(a>b && a>c)
printf(" the greatest number is a");
else if (b>a && b>c)
printf("the greatest number is b");
else
printf("the greatest number is c");
  getch();
}.
```

9. WAP to find out whether the given 4 digit number (year) is a leap year or not.

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
int y;
printf(" enter a year=");
scanf("%d", &y);
if(y% 4 ==0 && y% 100 != 0)
printf(" This year is leap year");
 else
printf(" This year is not leap year");
    getch();
    }.
```

2. few other programs:

Q. WAP in C to enter the marks obtained by the students in five subject. Calculate the total, perkentage and display division by the student.

Percentage	Division
p>=75	Distinction
p>=60 and <75	first division
p>=45 and <60	Second division
p>=35 and <45	Third division
otherwise	failed

```
#include <stdio.h>
#include (conio.h)
  void main()
  {
  int E, N, M, S, CS, t, per;
printf("Enter the marks of subjects");
scanf("%d%d%d%d%d", &E, &N, &M, &S, &CS);
t = E+N+M+S+CS;
per = (t*100)/5;
if (per >=75)
printf(" Distinction")
else if (per>=60 && Per<75)
printf(" first division");
elseif (per>=45 && per<60)
printf(" second division");
elseif (per>=35 && per<45)
```

Mukesh Singh SMC

```
printf(" third division ");
else
printf(" You are failed ");
getch();
}.
```

Q. WAP to read average temperature of a day in fahrenheit to print:
"Nice day" if temp is >60 && <80
"Cold day" if temp is <=60
"Hot day" if temp is >=80.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int T;
printf("enter the temperature");
scanf("%d",&T);
if (T>60 && T<80)
printf(" Nice day");
else if (T<=60)
printf(" Cold day");
else if (T>=80)
printf(" Hot day ");
else
printf("No day");
getch();
}.
```

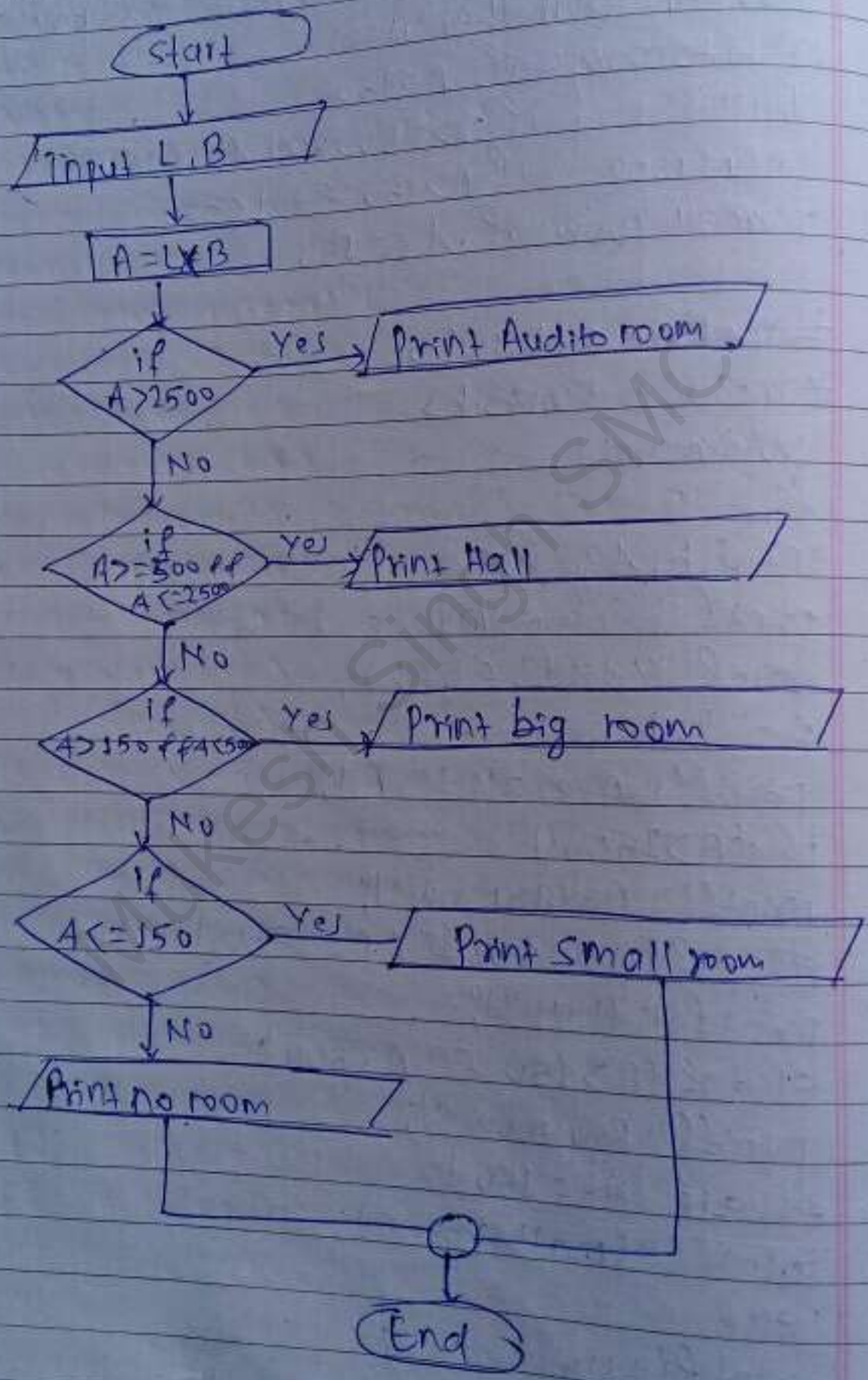
Mukesh Singh SMC

Q. Write a flowchart and C-program to read length and breadth of a room and print its area. Also find whether it is:
"Audito room" if A>2500
"Hall " if A>=500 && A<=2500
"Big room" if A>150 && A<500
"Small room" if A<=150

```
#include <stdio.h>
#include <conio.h>
void main()
{
int l,b,A;
printf("Enter values of l & b");
scanf("%d%d",&l,&b);
A = l*b;
printf("Area=%d",A);
if (A>2500)
printf("Audito room");
else if (A>=500 && A<=2500)
printf("Hall");
else if (A>150 && A<500)
printf("Big room");
elseif (A<=150)
printf("Small room");
else
printf("No room");
getch();
}
```

Mukesh Singh SMC

Flowchart



Mukesh Singh SMC

Page No.
Date

Q. WAP to input number and display it, if it is exactly divisible by 5 but not by 11.

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
int N;
printf("enter a number");
scanf("%d",&N);
if (N%5==0) && (N%11!=0)
printf(" Number is divisible by 5 but not by 11 ");
 else
printf(" Condition is not satisfied");
  getch();
 }
```

Q. Write a simple program to use Switch case.

```
#include <stdio.h>
#include <conio.h>
 void main()
  {
 int a;
printf(" press 1 or 2 \n");
scanf("%d", &a);
switch (a)
 {
```

Mukesh Singh GMC

```
case 1:
printf(" You have entered 1: ");
break;
case 2:
printf("You have entered 2!");
break;
default:
printf("Wrong input");
}
getch(); }
```

Q. Write a program to use switch case and display seven days individually

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a;
printf(" enter a number "); //OR printf(" press 1,2,3,4,5,6 or 7 ");
scanf("%d", &a);
switch(a)
{
case 1:
printf(" Sunday ");
break;
case 2:
printf("Monday");
```

Mukesh Singh SMC

```
 break;
 case 3:
 printf("Tuesday");
  break;
 case4:
 printf("Wednesday");
 break;
  case 5:
 printf("Thursday");
 break;
 case 6:
 printf(" Friday");
  break;
 case 7:
 printf("Saturday");
  break;
  case 8:
 printf(" invalid number");
  break;
  }
 getch();
 }
```

MukeshSingh SMC

Q. Write a program to read two numbers and display the following menu column using switch case.

Menu

1. Summation
2. Sum of squares
3. Sum of cubes
4. ~~Sum of~~ product
5. Exit

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
void main()
{
int n, s, p, x=5, y=4, cub, sqr, prod;
printf("\n\t case 1: Summation");
printf("\n\t case 2: Sum of cubes");
printf("\n\t case 3: Sum of squares");
printf("\n\t case 4: ~~Product~~ ~~Sum of~~ product");
printf("\n\t case 5: exit");
printf("\n\t ~~case~~ enter the case=");
scanf("%d", &n);
switch(n)
{
case 1:
s=x+y;
```

```
printf("Sum = %d", s);
break;
case 2:
cub = pow(x,3) + pow(y,3);
printf("Sum of cubes = %d", cub);
break;
case 3:
sqr = pow(x,2) + pow(y,2);
printf("Sum of squares = %d", sqr);
break;
case 4:
prod = x * y
printf("product = %d", prod);
break;
case 5:
exit(0);
~~break~~;
}
getch();
goto icte;
}
```

Mukesh Singh SME

1. WAP to display following hotel menu:

1. Samosa
2. Pizza
3. Burger
4. Jerry
5. Momo
6. Exit

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
void main()
{
int n;
menu:
printf("\n\t case 1: Samosa");
printf("\n\t case 2: pizza");
printf("\n\t case 3: burger");
printf("\n\t Case 4: Jerry");
printf("\n\t case 5: momo");
printf("\n\t case 6: exit");
printf("\n\t enter the case=");
scanf("%d", &n);
Switch(n)
{
Case 1:
printf(" Samosa");
break;
```

Mukesh Singh SMC

```
case 2:
printf("pizza");
break;
case 3:
printf("burger");
break;
case 4:
printf("jerry");
break;
case 5:
printf("momo");
break;
case 6:
exit(0);
}
getch();
goto menu;
}
```

OR Another method

```
#include <stdio.h>
#include <conio.h>
void main()
{
menu:
int a;
printf("press 1,2,3,4,5\n");
scanf("%d", &a);
```

Mukesh Singh SMC

```
switch(a)
{
case 1:
printf(" samosa per plate Rs 30");
break;
case 2:
printf(" pizza per plate Rs 50");
break;
case 3:
printf(" burger per plate Rs 50");
break;
case 4:
printf(" jerry per plate Rs 30");
break;
case 5:
printf(" momo per plate Rs 50");
break;
default:
printf(" thanks for visiting");
}
getch();
goto menu;
}
```

Mukesh Singh SMC

Page No.
Date

3.1.5 Conditional Operators (?:)

The operator pair "?:" is known as conditional operator. Unlike all other operators, ?: in C operator is ternary operator i.e. it takes three operands. The conditional operator has the following construct.

expr1 ? expr2 : expr3

Here, expr1 is evaluated first. If expr1 is true, the value of expr2 is the value of conditional expression. If expr1 is false, the value of expr3 is the value of conditional expression.

```
#include <stdio.h>
#include <conio.h>
int main()
{
int x=15,y;
y=(x>5)?3:5;
printf("%d",y);
getch();}
```

3.2 Looping Structure (while, do while and for):

~~Looping or~~ Loop is a control structure that executes the same program statement or block of program statements repeatedly for specified number of times or till the given condition is satisfied. Loop structure is used to carry out looping.

For example, if we want to display

Mukesh Singh SMC

"C is the best" 10 times, one way to get the desired output is - we type printf("C is the best"); 10 times, which is time consuming and hence not preferable. Another way to perform this is - use a loop structure. With loop structure we don't need to type the same program statement again and again. Loop structure is also known as repetitive control structure or iterative control structure.

Example:

Q. Write a program to display the message "Welcome to my College" ten times:

a) Without using loop:

```
#include <stdio.h>
# include <conio.h>
  void main()
  {
//int i;
Clrscr();
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
Printf("\n Welcome to my College");
printf("\n Welcome to my College");
```

```
printf("\n Welcome to my College");
 getch(); }
```

b) Using loop:

```
#include <stdio.h>
#include <conio.h>
 void main()
 {
 int i;
 clrscr();
for (i=0; i<10; i++)
printf("\n Welcome to my College");
 getch();
 }
```

Types of loop

Mainly there are three types of loop: while, do while, for loop

3.2.1. While Loop:

It executes the program statements repeatedly until the given condition is true. It checks condition at first; if it is found true then it executes the statements written in its body part (inside parenthesis) otherwise it just gets out from the loop structure. It is also known as entry control or pre-test loop.

Mukesh Singh SMC

Algorithm Syntax:

initialization
while (condition)
{
statements;
increment/decrement;
}

Where initialization means starting point, condition means stopping point and increment/decrement means counter.

Flowchart

3.2.2 Do-while Loop

It also executes program statements repeatedly until the given condition is true. It executes the program statements once at first then only condition is checked. If condition is found true then it executes the program statements again, otherwise it gets out from the loop structure. As it checks condition at last it is also known as post-test loop or exit control loop.

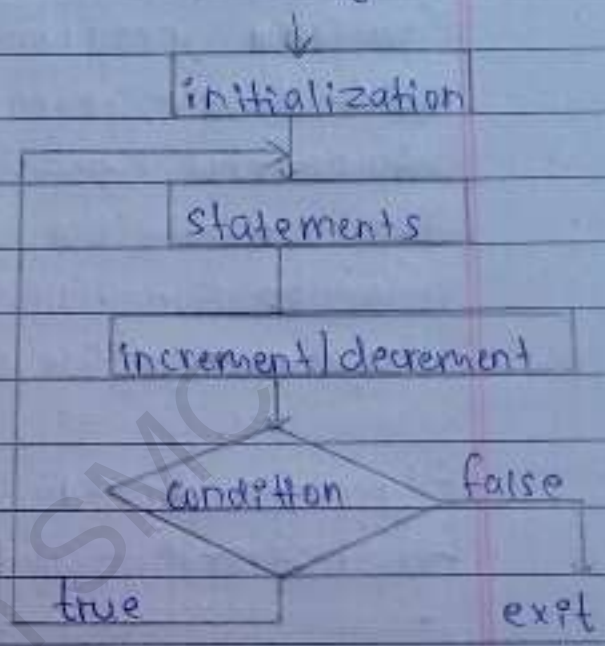
Mukesh Singh SMC

Page No.
Date

Algorithm Syntax

```
initialization
do
{
  statements;
  . . . . .
  increment / decrement;
} while (condition)
```

Flowchart



	While loop		Do while loop
1.	In while loop, condition is checked in the beginning.	1.	In do while loop, condition is checked at the end.
2.	It is also known as pre-test or ~~control~~ entry control loop.	2.	It is also known as post-test or exit control loop.
3.	It is not terminated with semicolon.	3.	It is terminated with semicolon.
4.	Statements are not executed if the condition is false.	4.	Statements are executed once even the condition is false.
5.	It uses keyword 'while'.	5.	It uses two keywords 'do' and 'while'.

Mukesh Singh SMC

3.2.3. For Loop:

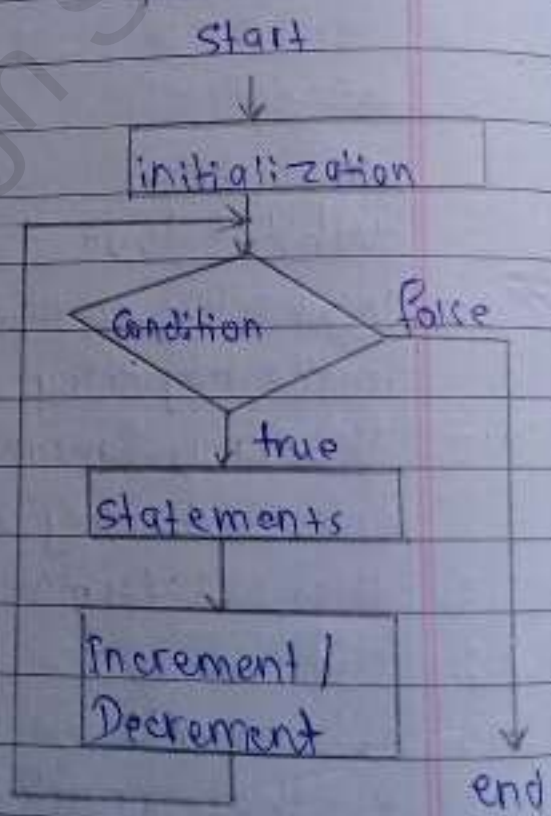
For loop is the most commonly used loop in any programming languages. It is used to execute program statements or block of program statements repeatedly for specified number of time. It is definite loop. It consists of three expressions: initialization, condition and counter which are defined within a statement.

Algorithm syntax

```
for (initialization; condition; counter)
{
statements;
}
```

Where, initialization is starting point, condition is stopping point and increment/decrement is counter.

Flowchart:



3.2.4. Nested Loops

We can write an entire loop structure inside another loop structure. So loop inside another loop is called nested loop.

Mukesh Singh SMC

Page No.
Date

Syntax

Nested for loop

```
for (initialization ; condition ; increment / decrement)
 {
 for (initialization ; condition ; increment / decrement)
 {
   Statements;
  }
}
```

Mukesh Singh SMC

Ex

Loop

for loop
while loop
do... while loop

} initialization
condition
increment / decrement

for loop

Mukesh Singh SMC

While loop.

```
<initialization>
 while ~~loop~~ <condition>
 {
 <statement>
 <increment / decrement>
    }
```

Example

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

Mukesh Singh SMC

* do-- while loop

Syntax:

```
<initialization>
  do
 {
 <statement(s)>;
increment /decrement;
  }
```

Example

Write a program to print natural numbers from 1 to 10 using do while loop.

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

Mukesh Singh GMC

}

Few programs for practice

Q. Write a program to read any integer and print its multiplication table (using for loop).

```
#include <stdio.h>
#include <conio.h>
  void main()
  {
  int a,b,c;
printf("Enter a number=");
scanf("%d",&a);
 for (b=1; b<=10; b++)
  {
  c= a*b;
printf("%d * %d= %d\n", a,b,c);
  }
 getch();
  }
```

Using while loop:

```
#include <stdio.h>
#include <conio.h>
  void main()
  {
 int a,b,c;
printf("Enter any number=");
```

Mukesh Singh SMC

```
scanf("%d", &a);
b=0;
while (b<10)
{
b++;
c= a*b;
printf("%d * %d = %d\n", a,b,c);
}
getch();
}
```

Using 'do... while loop'.

```
#include <stdio.h>
#include <conio.h>
void main().
{
int a,b,c;
printf("Enter any number=");
scanf("%d", &a);
a=1
do
{
a++;
c= b*a;
printf("%d * %d = %d\n", a,b,c);
}
while (a<=10);
getch();
}
```

Mukesh Singh SMC

Q. Write a program to print 1 to 10 natural numbers and sum them.

Using forloop:

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

Using while loop

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

Mukesh Singh SMC

Page No.
Date

Q. Write a program to print multiplication table from 1 to 10.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c;
for(a=1; a<=10; a++){
{
for (b=1; b<=10; b++)
{
c=a* b;
print("\t %d",c);
}
}
getch();
}
```

Mukesh Singh SMC

WAP for print factorial for input number using for loop.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, i; fact=1;
printf("Enter any number =");
scanf("%d", &n);
for (i=1; i<=n; i++)
{
fact = fact * i;
}
printf(" factorial=%d", fact);
getch();
}
```

WAP to print ASCII charactor from 1 to 128

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n;
for (n=1; n<=128; n++)
printf(" ascii number = %d, character = %c \n", n, n);
getch();
}
```

Mukesh Singh SMC

Q. WAP to input any number and print its reverse number.

```
#include <stdio.h>
#include <conio.h>
   void main()
   {
int n, reve = 0, rem;
printf(" Enter any number = ");
scanf ("%d", &n);
while (n != 0)
rem = n % 10;
reve = reve * 10 + rem;
n = n / 10;
 }
printf(" reve = %d", reve);
getch();
   }
```

Mukesh Singh SMC

Q. WAP to read 3 digit number and to test whether it is a Armstrong number or not.
$(a^3+b^3+c^3 = abc)$

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
int n,n1,d1,d2,d3,arm;
printf("Enter number= ");
scanf("%d",&n);
n1=n;
d1=n%10;
n=n/10;
d2=n%10;
n=n/10;
d3=n;
arm= pow(d1,3)+pow(d2,3)+pow(d3,3);
if (n1==arm)
printf("armstrong...");
else
printf("Not armstrong");
getch();
}
```

Mukesh Singh SMC

WAP to all three digit Armstrong number.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
int n1,n, d1, d2, d3, arm;
printf("Enter three armstrong numbers =");
for (n1=100; n1<=999 ;n1++)
{
n=n1;
d1 = n% 10;
n=n/10;
d2 = n% 10;
n= n/10;
d3 = n;
arm = pow(d1,3) + pow(d2,3) +pow(d3,3);
if (arm == n1)
printf("\n%d",n1);
}
```

Display the series : 1,5,9,13,17... n^{th} term.

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

Rajesh Singh SMC

```
int i, n, a;
printf("Enter the value of nth term");
scanf("%d", &n);
a=1;
for (i=1; i<=n; i++)
{
printf("%d\t", a);
a=a+4;
}
getch();
}
```

WAP to enter marks of five subjects and find out total marks.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, m, s=0;
clrscr();
printf("Enter student marks \n");
for (n=1; n<=5; n++)
{
scanf("%d", &m);
s=s+m;
}
printf("\n sum =%d", s);
getch();
}
```

Mukesh Singh SMC

WAP to 10 numbers and sorting in ascending.

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

WAP to ~~read 3 do~~ generate the following fibonacci series: 1, 1, 2, 3, 5, 8 ---- 25 terms.

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

Mukesh Singh SMC

```
int x, y, z, count;
  x=1;
  y=1;
  ~~printf("%d %d", x, y);~~
  printf("%d, %d", x, y);
  for (count=1; count<=25; count++)
   {
   z = x+y;
   printf("%d", z);
   x=y;
   y=z;
   }
   getch();
}
```

WAP to read 15 person's age and find out how many person fall under the following categories

Baby - age 0 to 5
Attending School - age 6 to 17
Adult - age 18 and over
(Using for loop)

```
#include <stdio.h>
#include <conio.h>
void main()
 {
 int baby=0; adult=0, school=0, count, age;
```

Mukesh Singh SMC

```
clrscr();
for (count=1; count<=15; count++)
{
printf("enter the age of our baby age:%d", count);
scanf("%d", &age);
if(age>=0 && age<=5)
   baby = baby+1;
else if(age>=6 && age<=17)
   school = school+1
else
   adult = adult+1;
}
printf("number of baby=%d", baby);
printf("\nnumber of school age=%d", school);
printf("\n number of adult age =%d", adult);
getch();
}
```

WAP to read 15 persons age maximum, minimum.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int max=0, min=100, count, age;
clrscr();
for(count=1; count<=15; count++)
{
```

Mukesh Singh SMC

```
printf("Enter age of 1st student =");
scanf("%d", &age);
if(age >= max)
max = age;
if(age <= min)
min = age;
}
printf("\n max age = %d", max);
printf("\n min age = %d", min);
getch();
}
```

WAP to find out sum of a given ~~digit~~ integer of arbitrary length.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, sum = 0, d;
clrscr();
printf("Enter the integer =");
scanf("%d", &n);
while (n != 0)
{
d = n % 10;
sum = sum + d;
n = n / 10;
```

Mukesh Singh SMC

```
}
printf("value of arbitrary length =%d", sum);
getch();
}
```

WAP to show this:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
int a,b;
for(a=1; a<=5; a++)
{
printf("\n");
for(b=1; b<=a; b++)
printf("%d", a);
}
getch();
}
```

Mukesh Singh SMC

WAP to show this:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
~~int~~ clrscr();
int a,b;
for(a=1; a<=5; a++)
{
printf("\n");
for(b=1; b<=a; b++)
printf("%d", b);
}
getch();
}
```

WAP to generate the following series and pri
sum : 1*4, 2*7, 3*10 ... n terms*1

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

Mukesh Singh SMC

```
{
int a = 1, b=4, y, z, n, sum=0;
 clrscr();
 a=1;
 b=4;
 printf(" enter the y= ");
 scanf("%d", &y);
for (n=1; n<=y; n++)
 {
 z = a* b;
 sum = sum+z;
 printf("\n series : %d x %d \t ", a, b);
  b=b+3 ;
 }
 printf("\n total value of series : %d ", sum);
 getch();
 }
```

WAP to generate the following series : 1*4, 2*7, 3*10... n terms.

```
# include <stdio.h>
#include <conio.h>
  void main()
  {
  clrscr();
  int n=1; x=4, y, z;
  printf("enter the y= ");
```

Mukesh Singh SMC

```
scanf("%d", &y);
for(n=1; n<=y; n++)
{
z=n*x;
x=x+3;
printf("\n series: %d x %d \t", n, x);
}
getch();
}
```

WAP to print 50 prime numbers:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, j;
clrscr();
for (i=2; i<=200; i++)
{
for (j=2; j<=i-1; j++)
if (i%j==0)
break;
if (i==j)
printf("%d\t", i);
getch();
}
```

Mukesh Singh SMC

Unit - 4

Function

4.1. Function Concept:

A function is a self-contained block of code that performs a particular task.

Functions are sub programs. They perform well defined tasks. They may be developed separately and tested. In other words, a function is a self-contained block of statements that perform a coherent (relational) or specific task of some kind.

Every C-program can be thought of as a collection of these functions. It is something like hiring a person to do a specific job for us. Sometimes, the interaction with this person is very simple, & Sometimes its complex.

Advantages of function:

- Function increase code reuseability.
- The length of function can be reduced by using functions at appropriate places.
- The program development will be faster.
- The program debugging will be easier.
- Large number of programmer can be involved.
- The program can be developed in short period of time.
- The program can be developed in different places.

4.2. Function Prototype, Call and definition:

There are three components of function:

(a) Function prototype

(b) Call the function

~~Function~~ (c) Function definition

→ (a) Function prototype:

Function prototype provides the following information to the computer:

- The type of the value returned.
- The name of the function
- The number and the type of the arguments that must be supplied in a function call.

Syntax

return-value function-name (arg 1, arg 2 ... argn)

Example:

int sum(int, int);

In this example, two arguments int types are passed inside function name 'sum' and it returns int value.

Prototype only needed if definition comes after use in program. They are generally written at the beginning of a program, ahead of main() function.

(b) Call the function or function call:

A function call is specified by the function name followed by the arguments enclosed in parenthesis and terminated by a semicolon.

Syntax:

variable = function-name (arg 1, arg 2, ..., argn);

or, function-name (arg 1, arg 2, ..., argn);

Where passing arguments arg1, arg2, arg3 ... are optional.

Mukesh Singh SMC

Example:

i) c = sum(a, b)

ii) area(r);

Calling function	Called function
main()	// calling function
{--------	
function();	// function call

}	
function()	// called function (body of the function)
{-----	
----	
statements	

}	

In the above example, the function() is called by the main() function and the code of the function (body of the function) is written after the main() function.

Example to print square of a number using function:

```
#include <stdio.h>
#include <conio.h>      // function prototype
int value(int);         // calling function
void main()
{
int a, b;
printf("\nEnter a number");
```

```
scanf("%d", &a);
b = value (a)          // call function
}
int value (int a)      // function declarator
{
int p;
p = a*a;               } function body (called function)
return (p);
}
```

(c) Function definition:

A function definition has two principle components:

#. Function header or function declarator
#. Body of the function

The first line of function definition is known as function declarator and is followed by the function body. Function declarator contains the value returned by the function followed by the function name and optionally a set of arguments separated by commas and enclosed in parenthesis. Each argument is preceded by its associated type declaration. An empty pair of parenthesis must follow the function name if the function definition does not include any arguments.

Mukesh Singh SMC

Syntax:

data_type function-name(datatype arg1, data type arg2, --- data type argn).

where data-type represents the data type of the item that is returned by the function, function-name represents the function name and arg1, arg2, arg3 --- argn are variables (for arguments).

Example:

```
int sum(int a, int b) //  function declarator
{
int c;
c = a*b;
return(c);
}
```

} function body

4.3. Different ways of Using function (Types of function):

→ Functions are of two types:

A. Library (built-in) function

B. User defined function

A. Library function:

C has the facility to provide some library function to the programmer for doing some operations. As an example C has a mathematical function which is used for finding out the square root of any number, String library functions are used for string operations, as example strlen() is used to find out the length of the string and strcmp() is used to

for comparison of two strings. These operations are programmed and stored as library functions, so that they can be called by any program.

Mathematical functions are called by using the preprocessor directive #include<math.h> and string library functions are called by #include <string.h>

Example:

Program to find the square root of any number.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
int n;
printf("Enter the number:");
scanf("%d", &n);
printf("\n The square root of %d is %f", n, sqrt(n));
} // sqrt() library function
```

B User-defined function:

Library functions provide the facility for doing some predefined operations. User can create functions for doing any specific task of the program. Such functions are called user-defined functions.

User-defined function can be used (categorized) in four different ways:

1. No return and no parameter passing - Void add();
2. Return value and passing parameter - int add(int x, int y)
3. Return value and no parameter passing - int add()
4. No return and passing parameter - Void add(int x, int y)

1. No return and no parameter passing (function returning no value and passing no ~~parameter~~ arguments):

Here, an argument is not passed from the calling function to the called function and there is no return statement in the called function.

E.g:

```
#include <stdio.h>
#include <conio.h>
void sum();  //function prototype
void main()
{
 sum();
}
int sum()
{
int a,b,c;
printf("\nEnter two numbers");
scanf("%d%d", &a, &b);
c=a+b;
printf("\n sum is %d", c);
}
```

Mukesh Singh SMC

2. Return value and passing parameter
OR (OR function returning value and passing arguments):

→ Here, an argument is passed from the calling function to the called function and there will also be return statement in the called function.

Eg:

```
#include <stdio.h>
#include <conio.h>
~~void main()~~ int sum(int, int); // function prototype
void main()
{
int a, f;
printf("\nEnter two numbers");
scanf("%d %d", &a, &b);
f=sum (a, b);
printf("\n sum is %d", f);
}
int sum (int a, int b)
{
int c;
c=a+b;
return(c);
}
```

Mukesh Singh SMC

3. Return value and no ~~pass~~ parameter passing:
OR (function returning value and passing no arguments):
Here, an argument is not passed from the calling function to the called function. but there is return statement in the called function.

Eg:

```
#include <stdio.h>
#include <conio.h>
 int sum();  // function prototype
 void sum()
 {
 clrscr();
 int f;
 f= sum;
 printf("\nSum is %d", f);
 }

void sum()
 {
int a,b,c;
printf("\nEnter two numbers");
scanf("%d%d", &a, &b);
 c=a+b;
 return(c);
}
```

Mukesh Singh SMC

4. No return value and passing parameter:
or (Function returning no value but passing arguments):
Here, an argument is passed from the calling function to the called function but there is no need of return statement in the called function.

E.g:

```
#include <stdio.h>
#include <conio.h>
void sum (int, int); // function prototype
void main()
{
int a, b f;
printf("\n Enter two numbers:");
scanf("%d %d", &a, &b);
sum(a, b);
}
void sum (int a, int b)
{
int c;
c = a + b;
printf("\n sum is %d", c);
}
```

Mukesh Singh SMS

Difference between library function and user-defined function:

Library function	User defined function
1. Library functions are predefined functions.	1. User defined functions are functions which are created by the programmer according to the need.
2. Library function requires header file to use it.	2. User defined function requires function prototype to use it.
3. Program development time is faster.	3. Program development time is slower than library function.
4. It is called at run time.	4. It is called at compile time.
5. The name of library function can't be changed.	5. The name of user defined function can be changed at any time.
6. The program using library function will be usually short.	6. The program using user defined function will be usually lengthy.
7. Examples printf(), scanf(), strupr(), strlen()	7. Examples: day(), product(), sum(), diff() etc.

Mukesh Singh SMC

4.5 Recursion

Recursion can be regarded as the ability of function depending on object in terms of a simpler case of itself. It is a process by which a function calls itself repeatedly until some specific condition has been satisfied. For the problem to be solve recursively two conditions must be satisfied.

The function should call itself.

The problem statement must include a stopping condition.

E.g.

WAP to calculate sum of n natural numbers using recursion.

```
#include <stdio.h>
#include <conio.h>
int sum(int);
void main()
{
int n, f;
printf("Enter last number");
scanf("%d", &n);
f= sum (n)
printf("sum is %d", f);
}
int sum (int n)
{
if (n<=0)
return 0;
```

Mukesh Singh SMC

```
else
  return (n+sum (n-1));
  }
```

WAP to calculate factorial using recursive function.

```
#include <stdio.h>
#include <conio.h>
  ~~void main()~~ int fact(int);
void main()
  {
int a, f;
printf("\n Enter a number");
scanf("%d", &n);
f= fact(n);
printf("\n factorial of %d is %d", n, f);
}
int fact(int n)
{
if (n<=1)
 return(1);
else
 return (n* fact (n-1));
}
```

From the above program of factorial, the factorial of a number n is expressed as a series of repititive multiplication as:

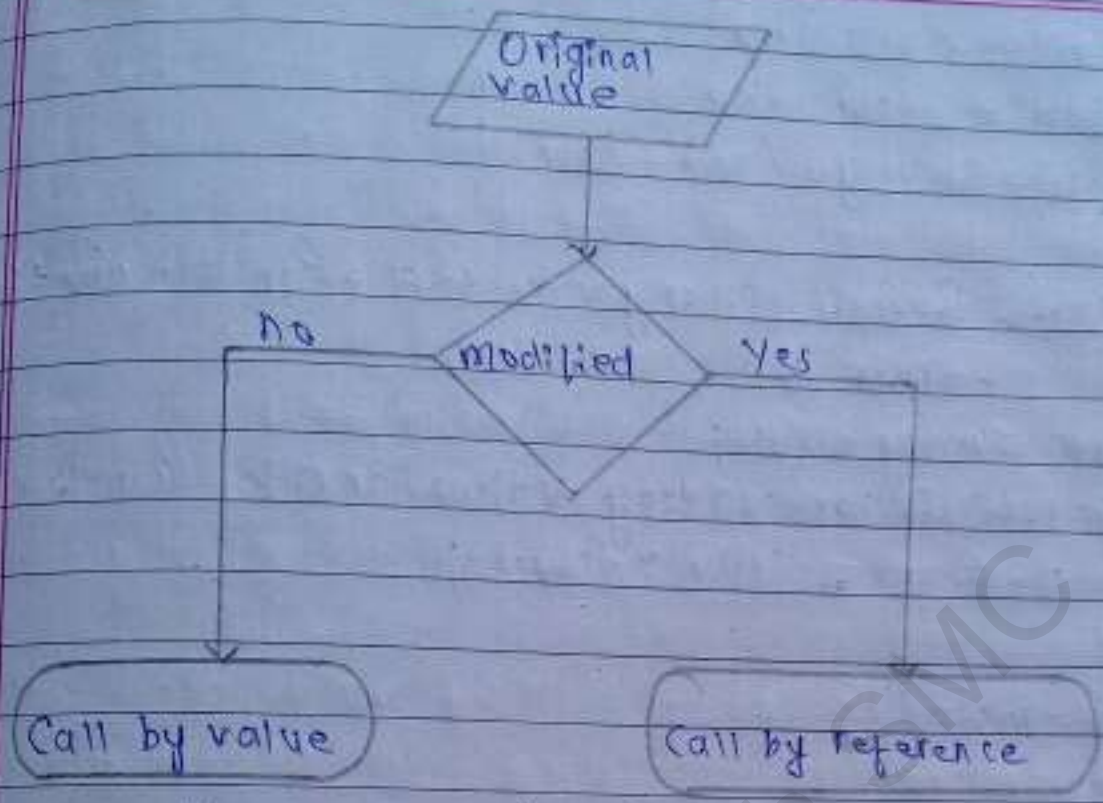
factorial of n = n(n-1)(n-2)(n-3)....1

Example:
factorial of 3 = 3 * 2 * 1 = 6
Let us see how the recursion works. Assume
n = 3. Since the value of n is not less than 1,
the statement
fact = n * fact(n-1);
will be executed with n = 3. That is,
fact = 3 * fact(2);
will be evaluated. The expression on the right
hand side includes a call to fact with n = 2.
This will return the following value:
2 * fact(1);

Once again, fact is called with n = 1. This time,
the function returns 1. The sequence of operations
can be summarized as follows:

fact = 3 * fact(2)
= 3 * 2 * fact(1)
= 3 * 2 * 1
= 6.

9.4 Call by value, call by reference:
→ There are two ways to pass value
or data to function in C language:
'Call by value' and 'Call by reference'.
Original value is not modified in call by
value but it is modified in call by reference.



Let's understand call by value and call by reference in C language one by one:

1. Call by Value:

In call by value, original value is not modified. In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If we change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Example of call by value is given in the next page:

Mukesh Singh SMC

```
#include <stdio.h>
#include <conio.h>
void change (int num)
{
printf("Before adding value inside function num=%d\n",
        num);
num = num + 100;
printf("After adding value inside function
        num = %d\n", num);
}
int main()
{
int x = 100;
clrscr();
printf("Before function call x = %d\n", x);
change (x); // passing value in function
printf("After function call x = %d\n", x);
getch();
return 0;
}
```

Output

```
Before function call x = 100
Before adding value inside function
After adding value inside function
After function call x = 100
```

Mukesh Singh SMC

2. Call by reference in C:

→ In call by reference, original values modified because we pass reference (address).

Here, address of the values is passed in the function, so actual and formal arguments shares the same address space. Hence value changed inside the function, is reflected inside as well as outside the function.

> Note: To understand the call by reference, you must have the knowledge of pointers.

Example of call by reference:

```
#include <stdio.h>
#include <conio.h>
 void change(int*num)
 {
 printf("Before adding value inside function num=%d\n",
          *num);
 (*num)=(*num)+100;
printf("After adding value inside function num=%d\n",
          *num);
 }
 int main()
 int x=100;
 clrscr();
printf("Before function call x=%d\n", x);
  change(&x); // passing reference in function
printf("After function call x=%d\n", x);
```

```
getch();
return 0;
}
```

Output:

```
Before function call x = 100
Before adding value inside function
After adding value inside function
After function call x = 200
```

→ Difference between call by value and call by reference in C:

Call by value	Call by reference
1. A copy of value is passed to the function.	1. An address of value is passed to the function.
2. Original value is not modified in it.	2. Original value is modified because we pass reference. (address)
3. Changes made inside the function is not reflected on other functions.	3. Changes made inside the function is reflected outside the function also.
4. Actual and formal arguments will be created in different memory location.	4. Actual and formal arguments will be created in same memory location.

Mukesh Singh SMC

Unit-5

Array, Pointer & String

Page No.
Date 1/12

5.1. Concept of Array:

An array is a collection of similar type of data items treated as single unit. An array is used to store a collection of data, but it is more often useful to think of array as a collection of variables of the same type. It acts to store data under the same name with an index, also known as subscript which helps to access individual array element. Array data type may be: int, float, char etc, depending upon the nature of problems.

It supports the homogeneous data type not heterogeneous. Arrays are widely ~~used in~~ within programming for different purposes such as sorting, searching, ~~etc.~~ indexing etc.

(Extra) Characteristics of Array:

- All the array elements share the common name.
- The elements of array are stored in contiguous memory locations.
- ~~By declaring or with~~
- It always starts from 0 (zero) position and ends at size -1.

e.g. int a[5] = {21, 23, 14, 15, 22}

21	23	14	15	22
0	1	2	3	4

Therefor, in an array with "n" elements, first index is '0' and the last index is 'n-1'.

- The value of each element of array is listed within two curly brackets { } and a comma (,) is used to separate one value from another.
- Array name must not be separate from the square brackets containing the index.

Syntax:-

data-type array-name[array size]={define variable};

int a[5] = {11,12,13,14,15} // right declaration

int a [5] = {11,12,13,14, 15} // wrong declaration

Extra

Advantages of array

- It is easier for handling similar types of data in a program.
- It is efficient for solving problems like sorting, searching, indexing etc.
- It is very close to matrix, therefore it is easy for solving matrix related problems.
- Graphics is an array of pixels so, graphics manipulation can be done easily using array.

Extra

Disadvantages:

- It is not possible to hold dissimilar type of data in array.
- It is difficult to visualize the multi-dimensional array.
- It is static in nature so it is difficult to define the size of array during running time.

Date

5.2 Array declare, access and initialization:

An array is a collection of data that holds fixed number of values of same type.

1. Array declare:

data-type array-name [array_size];

for example:

float mark[5];

Here, we declared ~~the~~ an array 'mark', of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

2. Access:

We can access elements of an array by indices.

Suppose we declared an array 'mark' as above. The first element is mark[0], second element is mark[1] and so on.

mark[0]	mark[1]	mark[2]	mark[3]	mark[5]

Few key notes:

- Arrays have 0 as the first index not 1. In this example, mark[0].
- If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4]
- Suppose the starting address of mark[0] is 2120d. Then, the next address ~~no~~ a[1], will be 2124d

Mukesh Singh SMC

address of a[2] will be 2128d and so on. It is because of the size of float is 4 bytes.

3. Initialization to an Array:

It is possible to initialize an array during declaration. for example;

int mark[5] = {19, 10, 8, 17, 9}

Another method to initialize array during declaration

int mark[ ] = {19, 10, 8, 17, 9};

mark[0]	mark[1]	mark[2]	mark[3]	mark[4]
19	10	8	17	9

Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9.

Mukesh Singh SMC

Types of Array:

There are ~~two~~ two types of array which are as follows:

1. One dimensional Array
2. Two dimensional Array / Multi dimensional Array

1 One dimensional Array:

An array which has only one subscript is named as one dimensional array. A subscript is a number of large bracket in which we put the size of the array.

2. ~~(5.3) Multi dimensional Array~~:

Declaration of One dimensional array:

data_type array-name. [array-size];

For example: int age[5];

Here, the array name is age. The size of array is 5, i.e., there are 5 items (elements) of the same type (int, in this case).

int age[5];

age[0]	age[1]	age[2]	age[3]	age[4]

2 (5.3) Two/Multi-dimensional Array:

Multidimensional arrays are defined the same as one-dimensional array, except that it consists more than one pair of square

Mukesh Sir SMS

bracket that is subscript. Thus a two-dimensional array will require two square brackets, one for row size and other for column size. The maximum capacity of elements of array is the product of row size and column size. A three dimensional array will require three square brackets and so on.

Syntax:

data_type array-name[row-size][column-size];

eg: int a[2][2]

We can represent matrix in double dimensional array.

2	3
1	4

int a[2][2] where

A[0][0] = 2

A[0][1] = 3

A[1][0] = 1

A[1][1] = 4

In two dimensional array (suppose 3x3 matrix), data items are stored in memory below:

	Column 0	Column 1	Column 2
	[0][0]	[0][1]	[0][2]
Row 0	100	200	300
	[1][0]	[1][1]	[1][2]
Row 1	200	3150	50
	[2][0]	[2][1]	[2][2]
Row 2	350	500	600

As in the above example, two dimensional array consist 3 rows and 3 columns, the starting array element is 100 that is stored in position matrix[0][0] and last array element is 600 that ~~consist~~ is stored in position matrix[2][2].

This means,

```
int matrix[3][3] = {100,200,300,200,150,50,350
                    ,500,600};

matrix[0][0] = 100;
matrix[0][1] = 200;
matrix[0][2] = 300;
matrix[1][0] = 200;
matrix[1][1] = 3150;
matrix[1][2] = 50;
matrix[2][0] = 350;
matrix[2][1] = 500;
matrix[2][2] = 600;
```

~~matrix[2~~

Mukesh Singh SVC

~~5.4~~ ~~Concept of Pointer~~ :

One dimensional array	Two dimensional array
1. It consists of only one subscript.	1. It consists of two subscript.
2. Maximum size will be the size of array which we define in program.	2. Maximum size will be the product of row and column size of array which we define in program.
3. It stores data either row wise or column wise.	3. It stores data in matrix form i.e. row wise and column wise both.
4. Syntax: data_type array_name[array size];	4. Syntax: data_type array_name [row size][column size];
5. Example: int num[5];	5. Example: int matrix[2][2];
6. Program:	6. Program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, num[5];
printf("\nEnter elements:");
for(i=0; i<5; i++)
  scanf("%d", &num[i]);
printf("\nArray elements are:");
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, j;
int matrix[2][2]={1,2,3,4};
printf("\nThe 2x3 matrix is:");
for (i=0; i<2; i++)
{
```

```
for (i=0; i<5; i++)
  printf("%d\t", num[i]);
}
```

```
for (j=0; j<3; j++)
printf("%d\t", matrix[i][j]);
printf("\n");
}
}
```

5.4. Concept of Pointer:

Pointers are variables that hold address of another variable of same data type.

Or, A pointer is a special type of variable that represents memory address of a variable rather than its value. Like other variables, it also has a name, has to be declared and occupies some space in memory. It is called pointer because it points to a particular location in memory (by storing the address of that location).

Pointer is one of the most powerful and useful feature of the C language, which distinguishes it from other languages. The concept of pointer is somewhat complex but it is simple when compared to its advantages. Pointer is more difficult to understand but the proper use of it makes the C language excellent.

Features of Pointer:

- Pointer saves memory space.
- The execution time is faster because data manipulation is done directly inside memory.

- Pointer is closed to array so it is efficient for solving array and string related problems.
- Pointer is close to hardware so it is very efficient for solving hardware interfacing problems.

5.5. Pointer address, dereference, declaration, assignment, initialization:

1. Pointer address, dereference:

The unary operator '&' is the address operator, it represents the "address of the variable" (e.g. &a).

And the indirection or dereference operator '*' is the value at address operator. It represents the value at the specified address. (eg. *a)

Example:

```
int *p = &q;
```

In the above example p returns the address of q to the variable p or we can simply say that p is a variable which contains the address of an integer variable q. We can also declare as - int *p;

In the above example, p is a pointer variable, we use * for pointer variable.

Page No.
Date

Dereferencing pointer:

Dereferencing is just indirection. It is telling the compiler, "I have the address of a variable in the pointer. I want to access that pointed-to address either to get a value or set a value". A pointer holds a reference to a variable; the reference being the memory address stored in pointer. When we access the value at that reference, we de-reference the pointer.

Dereferencing can be used to either indirectly get a value from the pointer address or to assign a value to the pointer address.

Let's look at an example:

```
#include <stdio.h>
#include <conio.h>
int main (int argc, char ** argv)
{
// declare int ival and int *iptr
int ival = 1;
int *iptr = &ival;

// dereference iptr to get v
  int get = *iptr;
  printf("*iptr = %d\n", get);

  // dereference iptr to set v
  *iptr = 2;
  int set = *iptr;
```

```
printf("*iptr = %d\n", set);
printf("ival = %d\n", ival);
}
```

Declaration of a Pointer:

We can declare ~~the~~ a pointer variable likewise the normal variable declaration by putting the * operator before variable.

Syntax:

```
Data_type * variable_name;
```

Example

```
char *p;
float *q;
```

→ char *p means 'p' contains the address of the variable which is the character type variable.

→ float *q means 'q' contains the address of the variable which is of floating type.

Pointer Assignment:

This is a type of expression used to assign value of one pointer to another using assignment operator '='. In this, value of right hand side, points to memory location of variable stored in left hand side pointer. As a result both pointers points the same memory location after this expression.

For eg: assignment y = x; makes y point to the same pointee as x.

Pointer initialization:

Pointer initialization is the process of assigning address of a variable to pointer variable. Pointer variable contains address of variable of same data type. In C language, address operator '&' is used to determine the address of variable. The '&' (immediately preceding a variable name) returns the address of the variable associated with it.

```
int a=10;
int * ptr;    // pointer
declaration
ptr = &a;     // pointer
initialization
or,
int * ptr = &a; // initialization and declaration
                   together.
```

5.6. Pointer Arithmetic:

We can perform arithmetic operations on pointer variable just as we can a numeric value. As we know that, a pointer in C is a variable which is used to store the memory address which is a numeric value. The arithmetic operations on pointer variable effects the memory address pointed by pointer.

Mukesh Singh SMC

Valid pointer arithmetic Operations:

- Adding a number on pointer
- Subtracting a number from a pointer.
- Incrementing a pointer.
- Decrementing a pointer
- Subtracting two pointers.
- Comparison on two pointers.

Invalid pointer Arithmetic Operations:

- Adding of two pointers.
- Division of two pointers.
- Multiplication of two pointers.

C program to show 'pointer arithmetic':

```
#include <stdio.h>
#include <conio.h>
void main()
{
int int_val = 10, *int_ptr
char char_val = 'A', *char_ptr
float float_val = 4.65, *float_ptr

/* initialization pointers */
int_ptr = &int_val;
char_ptr = &char_val;
float_ptr = &float_val;

printf("Address of int_val
```

Mukesh Singh SMC

```
printf(" Address of char_var
printf(" Address of float_val

/* Incrementing pointers */
int_ptr++;
char_ptr++;
float_ptr++;
printf(" After increment address in int_ptr=%d",
printf(" After increment address in char_ptr=%d",
printf(" After increment address in float_ptr=%d",

/* Adding 2 to pointers */
int_ptr = int_ptr + 2;
char_ptr = char_ptr + 2;
float_ptr = float_ptr + 2;

printf(" After addition address in int_ptr=%d",
printf(" After addition address in char_ptr=%d",
printf(" After addition address in float_ptr=%d",

getch();
return 0;
}
```

Output

Address of int_var = 2293300
Address of char_var = 2293299
Address of float_val = 2293292

Mukesh Singh SMC

After increment address in int_ptr = 2293304
After increment address in char_ptr = 2293300
After increment address in float_ptr = 2293296

After addition address in int_ptr = 2293312
After addition address in char_ptr = 2293302
After addition address in float_ptr = 2293304

Mukesh Singh SMC

Date 1/13

5.7. Array and Pointer:

```
int *p;
int x[5] = {5,10,15,20,25};
```

Array and pointer in C are very closely related. They are so similar to each other in nature that they can be used interchangeably in each other position most of the time.

If 'a' is one dimensional array, then the address of the first element can be written as &a[0] or simply a. Morever, the address of the second array element can be written as a[1] or simply (a+1). In general, address of array element (i+1) can be expressed as either &a[i] or as (a+i). The expression (a+i) is a symbolic representation for an address rather than an arithmetic expression. So, array a[i] and *(a+1) both represent the contents of that address that is the value of ith element of a.

Important link joining them is that array name without the brackets is the pointer name and other end a pointer can be indexed as if it's an array.

Q. WAP to show the relationship between array & pointer.

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

```
int i, num[50];
int *p1;
for(i=0; i<50; i++)
    num[i]=i;
p1 = num;  //Assigning base address of an array to pointer
for(i=0; i<50; i++) //Accessing array using index
printf("\n%d", p1[i]);
for(i=0; i<50; i++) //Accessing array using pointer arithmetic
printf("\n%d", *(p1+i));
}
```

Array	Pointer
~~1. An array is a single pre allocated chunk of contiguous~~	
1. An array is a collection of similar type of data items treated as single unit.	1. Pointer is a variable that ~~stores~~ hold address of another variable of same data type.
2. Array can be initialized at definition (eg int num[] = {2,4,5}.	2. Pointer can't be initialized at definition.
3. Array is static in nature.	3. Pointer is dynamic in nature.
4. Once memory is allocated, it cannot be resized or freed dynamically.	4. The memory allocation can be resized or freed later.
5. The assembly code of Array is different than pointer.	5. The assembly code of Pointer is different than Array

Mukesh Singh SMC

5.8 String:

String is the set of characters, digits and symbol. We can also define string as the array of characters. The end of string is marked with special character, the '\0' that means Null character. The size in character string represents the maximum number of characters that the string can hold.

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

Syntax:

```
char string_name[string_size];
```

As in above syntax, char refers to the character data type that will be stored in the array. string_name is the named string variable and string-size indicates number of characters that can be stored inside the array.

Example:

```
char name[10];
char address[10];
```

This declares the name as a character array (string) variable that can hold a maximum of 10 characters.

String is simply a sequence of characters, to declare a string we have to put data type char with string name with square brackets.

5.9. String functions in C:

The C library supports a large number of string handling functions that can be used to carry out many of the string manipulation. The header file #include <string.h> is used for string manipulation functions. Some of the common string manipulation functions are:

* strlen() function

The 'strlen() function' returns the length of a string which makes the string name as an argument.

Syntax

L = strlen(str);

Where str is the name of string and L is the length of string returned by strlen function.

* Strrev() function:

The strrev() function is used to reverse the given string.

Syntax:

Strrev(str);

Where str is string to be converted into upper case character.

* Strcpy() function

The strcpy() function is used to copy one string into other string.

Syntax: strcpy(str1, str2);

Mukesh Singh SMC

where str1, str2 are two strings. The content of string str2 is copied on string str1.

* strcat() function:

The strcat() function is used to join or ~~create~~ concatenate the string into other string.

Syntax:

strcat(str1, str2);

where str1, str2 are two strings. The content of string ~~is~~ str2 is copied on str1.

* strcmp() function:

The strcmp() function is used to compare two strings, character by character and stops comparison when there is a difference in the ASCII value or the end of any the string and returns ASCII difference of the characters that is integer.

Syntax:

v = strcmp(str1, str2);

where str1, str2 are two strings to be compared and v is returned value of differed characters.

* strlwr() function:

The strlwr() function is used to convert upper case characters of string into lower case characters.

Syntax:

strlwr(str);

where str is string to be converted into lower case characters.

* Strupr() function:
The strupr() function is used to convert lower case characters of string into upper case characters.

Syntax:
strupr(str);
where str is string to be converted into upper case characters.

Extra knowledge
Array of String:
A two dimensional array of character is called array of strings. It is also known as table of strings. It is very useful for solving string sorting problems.

5.10. Pointer and String:
A pointer may be defined as pointer to a character string

```
#include<stdio.h>
void main()
{
  char* text_pointer = "Good morning!!!
for(; *text_pointer != '\0'; ++text_pointer)
  printf("%c", *text_pointer);
}
```

Mukesh Singh SMC

Or another program illustrating pointer to text strings

```
#include <stdio.h>
void main()
{
char * days[] = {"Sunday", "Monday", "Tuesday",
        "Wednesday", "Thursday", "Friday", "Saturday"};
int i;
for (i=0; i<6; i++)
printf("%s\n", days[i]);
}
```

Remember that if the declaration is;

```
Char * pointer = "Sunday";
```

then the null character ('\0') is automatically appended to the end of the text string. This means that %s may be used in a printf statement, rather than using a for loop and %c to print out the contents of the pointer. The %s will print out all characters till it finds the null terminator.

Mukesh Singh SMC

Unit-6

Structure and Union

6.1. Concept of Structure:

Structure is collection of ~~homogeneous~~ heterogeneous (dissimilar) data items treated as a single unit. Each data item is called member of structure. The keyword 'struct' is used to declare structure variable and type of structure variable is defined by the tag-name of the structure.

Some major features of structure:

- It can be treated as record that is collection of interrelated data fields having different data types.
- It is possible to copy one structure with variable to another by simply assignment operator (=).
- It allows nested structure that is one structure inside another structure.
- It is possible to pass structure variable as parameter to any function.
- It is possible to define structure pointer also known as linked list.

Syntax of structure declaration:

In this example, member 1, member 2, ... member N are the members of the structure and var1, var2, var3 ... varM are M variable with data ~~ty~~ type tag-name.

```
struct tag-name
{
data-type member 1;
```

```
data-type member 2;
.....
data-type member N;
};
struct tage-name var1, var2, var3 ... varM;
```

Example

```
struct student
{
int roll-no;
char fname[30];
char lname[30];
}
struct structure S1, S2, S3;
```

6.2 Intitializing, accessing members of structure:

1. Structure initialization:

Structure variable can be initialized quite similar to array. The following example shows the structure initialization process.

```
struct student
{
int roll-no;
char fname[30];
char lname[30];
}
S1 = {1, "Ram", "Singh"};
```

Mukesh Singh SMC

The alternate method to initialize structure variable as:

```
struct student S2 = {1, "Sita", "Devi"};
struct student S3 = {2, "Hari", "Sharma"};
```

2. Accessing members of structure:

A member of structure can be accessed by using period (.) sign between structure variable and respective member.

Syntax:

variable.member

Example:

S1.roll_no S2.fname S3.lname

6.3. Array of Structure:

It is possible to declare structure as an array; called array of structure. It is also known as the collection of structure variables having same set of members. Consider the following structure variable S with data type student and array size 5.

Example:

```
struct student
{ int roll_no;
  char fname[30];
  char lname[30];
};
struct student S[5];
```

Mukesh Singh SMC

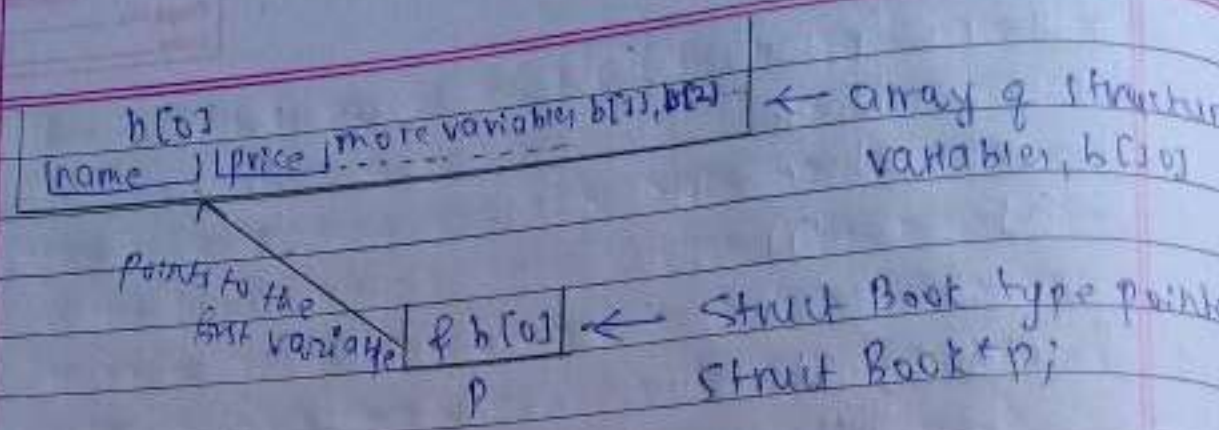
roll_no	roll_no	.....	roll_no
fname	fname		fname
lname	lname		lname
S[0]	S[1]		S[9]

6.4 Pointer to Structure:

Like we have array of integers, array of pointer etc, we can also have array of structure variables. And to make the use of array of structure variables efficient, we use pointer of structure type. We can also have pointer to a single structure variable, but it is mainly used with array of structure variables.

```
struct Book
{
char name[30];
int price;
}
int main()
{
struct Book a;   //single structure variable
struct Book* ptr; // pointer of structure type
ptr = &a;
struct Book b[30]; //Array of structure variables
struct Book* p;   //pointer of structure type
p = &b;
}.
```

Mukesh Singh SVC



6.5. Union:

Union is similar to ~~array~~ structure but it differs only in its storage location. The union keyword is used to define union. In structure, each member has its own memory ~~book~~ block whereas all members of union can share the same memory location. Therefore union can take less amount of memory than structure and it can access only one member at a time.

Syntax for declaration of union:

```
union tag-name
{
data-type member1;
data-type member2;
.....
data-type memberN;
};
union tag-name var1, var2, var3;
```

→ member1, member2.... memberN are the

members of the union and var1, var2 and var3 are variables with data type tag_name.

Example:

```
union_data
{
 char x;
 int y;
 float z;
}
var;
```

In this example, var is union variable with data type data. It contains three members: char x, int y and float z. In this example float z is the largest size member that requires 4 bytes memory and the same memory is shared over other two members.

6.6 Difference between union and Structure:

Structure	Union
1. It is the collection of data items having dissimilar data types.	1. It is same as structure but it differs in its storage class.
2. Memory size is determined by sum of memories allocated to all the members.	2. Memory size is determined by the memory allocated to the largest member.
3. Multiple members can be simultaneously access at a time.	3. Only one member can be accessed at a time.

Mukesh Singh SMC

4. The struct keyword is used to define structure.

5. Syntax

```
struct tag_name
{ data_type member1;
  data_type member2;
  ..............
  data_type memberN;
} var;
```

6. Example

```
struct data
{ char x;
  int y;
  float z;
} var;
```

7. Memory size for var is 7 byte.

4. The union keyword is used to define union.

5. Syntax:

```
union tag_name
{ data_type member1;
  data_type member2;
  ..............
  data type memberN;
} var;
```

6. Example:

```
union data
{ char x;
  int y;
  float z;
} var;
```

7. Memory size for var is 4 byte.

Mukesh Singh CMC

Page No.
Date

Differences between structure and Array:

Structure	Array
1. It is the collection of data items having dissimilar data types.	1. It is the collection of data items having similar data types.
2. Each data item is called member.	2. Each data item is called element.
3. It is user defined data type.	3. It is built in data type.
4. It is possible to define array of structure.	4. It is not possible to define array of array or structure of array.
5. Syntax: struct tag_name { data_type member1; data_type member2; ..... data_type memberN; } var;	5. Syntax: data_type array_name[size];
6. Example: struct temp { int p; float q; } x;	6. Example: int x[10];

Mukesh Singh SMC

Differences between function prototype and function definition:

Function prototype	Function definition
1. Function prototype provides type of value returned, name of function and number and type of arguments.	1. A function definition has three principle components: function header or function declaration and body of the function.
2. Syntax: return_value function_name (arg1, arg2, ... argn);	2. Syntax: data_type function_name (data_type arg1, data_type arg2, ... data_type argn)

3. Example: int sum (int, int);	3. Example: int sum(int a, int b) // function declaration
	{
	int c;
	c=a+b;
	return(c);
	} — function body

Mukesh Singh

Unit-7

Input Output and File Handling

7.1. Concept of Data file:

Many applications may require a large amount of data to be read, processed and also saved for later use. Such information is ~~used~~ stored on the auxiliary memory device in the form of file. A data file is simply a file in hard disk which is used to stored data permanently.

C language provides set of library functions for creating and processing data files. The prototype for all the file handling functions are declared in <stdio.h>.

7.1. Concept of file handling:

1. New way of dealing with data is file handling.
2. Data is stored onto the disk and can be retrieved whenever required.
3. Output of program may be stored onto the disk.
4. In C we have many functions that deals with file handling.
5. A file is a collection of bytes stored on a secondary storage device (generally a disk).
6. Collection of byte may be interpreted as -

- Single character
- Single Word
- Single line
- Complete Structure

7.2. File Access Methods:

Two types of file ~~accessing~~ processing techniques are used for accessing data file. They are:

1. Sequential File Processing
2. Random File Processing

1. Sequential File Processing:

In this type of technique, data is sequentially accessed from/to data file. It is very time consuming technique. If we need to access the last record of a file, then we need to access all the records before that record.

2. Random File Processing:

In this type of technique, data is randomly accessed anywhere from/to data file. It is more faster technique than the previous one. We can read and write data on particular part of file with the help of the functions ftell(), fseek() and rewind().

- ftell(): It gives current position in the file.
  n = ftell(fp);

- fseek(): sets the position to desired point.
  fseek(fp, offset, position);

- rewind(): set the position to the begining point.
rewind(fp);

7.3. functions of file handling: fopen(), fclose(), fflush(), freopen()

→ C language provides number of functions that helps to perform basic file operations. They are mentioned below:

- fopen(): Creates a new file or open a existing file
- fclose(): close data file
- fflush():
- freopen():

Few more functions:

- getc(): reads a character from a data file
- putc(): writes a character to a data file
- getw(): reads an integer from data file
- putw(): writes an integer to data file
- fscan(): reads formatted data from data file
- fprint(): writes formatted data to data file
- fread(): reads record from data file
- fwrite(): writes record to a data file

Mukesh Singh SMC

~~7.4. Formatted Input Output:~~

~~1. Formatted Input~~

7.4. Formatted input output (Formatted functions)

1. Formatted Input (scanf()):

Formatted input refers to an input data has been arranged in a particular format, for eg: 'int Ram', 'char Ram', 'float Ram' OR 50, Ram, 0.56. This is possible in C using the scanf(). The function scanf() is used for formatted input from standard input and provides many of the conversion facilities of the function printf().

Syntax:

```
scanf(format, num1, num2, ...);
```

eg:

```
scanf("%c%d", &Name, &Roll No);
```

2. Formatted Output:

Formatted output refers to the output of data that has been arranged in a particular format. The built-in function printf() is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output are the parameters to the printf() function.

Syntax: printf(format, data 1, data 2, ...);
~~print~~

Eg: printf ("%c", data 1);
~~print~~

7.5 Character Input Output (Unformatted Functions):

Unformatted functions do not allow the user to read or display data in desired format. These library functions basically deals with a single character or a string of characters. The functions: getchar(), putchar(), puts(), are considered as unformatted functions.

7.6 Direct Input/Output:

C provides additional file I/O library functions which allow direct input or output of the ~~library~~ binary, internal representation of data to files. This form of I/O is called block I/O, because data is transferred in blocks directly from the file to storage locations in memory with no conversion.

The library function provided for this type of Input/Output are fwrite() and fread().

7.7. Random Access file (Random file Access):
In this type of file processing, data is randomly accessed anywhere from/to data file. It is more faster technique than sequential file access. We can read or write data in particular part of a file with the help of the functions ftell(), fseek() and rewind().

- ftell(): It gives current position of the file in terms of bytes.
  n = ftell(fp);

- ~~fseek(): It takes file pointer as parameter and reset the position to the start of the file.~~
- fseek(): It helps to move the file position to the desired location within the file. It takes three parameters: fp-file pointer, offset position of the file in terms of bytes and position specifies three values: 0- beginning, 1- current position and 2- end of file.
  fseek(fp, offset, position);

- rewind(): It takes file pointer as parameter and reset the position to the start of the file.
  rewind(fp);

7.8. Error Handling:

As such, C programing does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code 'errno'. It is set as global variable and indicates an error occured during any function call. We can find various error codes defined in <error.h> header file.

The C programming language provides 'perror()' and 'strerror()' functions which can be used to display the text message associated with errno.

- The perror() function displays the string you pass to it, followed by a colon, a space, and then textual representation of the current errno value.

- The strerror() function, which returns a pointer to the textual representation of the current errno value.

7.9. File Operation:

In C, we can perform four major operations on the file, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing into a file.

1. Creating a new file: 2. Opening a file:

Let's suppose the file ~~newprogram~~ newprogram.txt doesn't exist in the location E:\program. The first function creates a new file named 'newprogram.txt' and opens it for writing as per the mode 'w'. The writing mode allows us to creat and edit the contents of file.

2. ~~Opening an existing file~~:

The fopen() function is used to create a new file or to open an existing file.

Syntax

```
*fp = FILE *fopen(const char *filename, const
                  char *mode);
```

Here filename is the name of file to be opened and mode specifies the purpose of opening the file. ~~Mode~~

*fp is the ~~file~~ FILE pointer (FILE *fp), which will hold the reference to the opened (or created) file.

Mode can be of following types:

Mode	description
r	opens an existing file for reading
w	opens a new file for writing
a	opens an existing file for appending
r+	opens an existing file for both reading & writing
w+	opens a new file for both reading & writing

a+ → open an existing file for both reading & appending

Few extras modes:

rb → opens a binary file for reading
wb → opens or creates a binary file for writing
ab → opens a binary file for appending
rb+ → opens a binary file for both reading & writing
wb+ → opens a binary file for both reading & writing
ab+ → opens a binary file for both reading and appending

3. Closing a file:

The fclose() function is used to close an already opened file.

Syntax:

int fclose(FILE * fp);

Here, fclose() function closes the file and returns zero on success, or EOF if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.

4. Reading from and writing to a file:

fprintf() function directly writes into the file, while fscanf() reads from the file, which can then be printed on console using standard printf() function.

Milresh Singh SMC