Fundamentals of "C" language

1:- What is c language?

Ans:- C is a computer programming language. C language replaced traditional programming languages of that time like: FORTRAN, PL/I and ALGOL. C was developed in 19## at AT and T's bell laboratories of USA, "Dennis Ritchie" is known as designer and writer of C language. C successfully combines the structure of HLL and the power and efficiency of assembly language or low level language.

2:- Features of c language:-

1:- C has a relatively easier syntax than FORTRAN, COBOL and PASCAL.

2:- C is an efficient and fast programming language. It is more than 50 times faster than BASIC.

3:- C language contains vast set of data types as compared to languages of its times.

4:- C language has 32 most commonly used keywords. In addition, 8 keywords are used with system programming.

5:- C language is a highly portable language. This means, if you have created a C program on one computer. It will run on the other computers without any modifications. The only exception may be a very different type of operating System that is not compatible with older OS in which program has

been created.

6.) C is a very well suited language for structured programming. The programmer can easily divide a problem into a number of modules or functions.

7.) C has got a rich set of library functions. These are the functions that provide ready-made functionality to users.

8.) C also has a support for graphics programming.

9.) C is an extendible language. This means users can add their own functions to the library set.

10.) Various versions of C language are available from various companies like Borland C Turbo C and ANSI C

11.) C can efficiently deal with bit, byte and word addresses.

3.) Programming Techniques :-

Programming in C is very similar to learning a language like hindi or English. We can understand the topic by below figures :-

Learning English

Alphabets ⇒ Words ⇒ Sentences ⇒ Paragraphs

Learning C programming :-

Alphabets Digit, Symbols	⟹	Constants Variables Keywords	⟹	Instructions	⟹	Program

—— Learning C language ——

4.] **Character Set :-**

A character can be any alphabet, digit or special Symbol Used to represent a piece of Information

Alphabets :- A B C ..... YZ
a b c ..... yz

Digits :- 0, 1, 2, 3 ..... 8, 9

Special Symbols :- ,, ~, ', @, #, %, ^, &, *, (,), _, -, +, =, /, \, |, {, }, [,], :, ;, ", ', <, >, ., ?, !

Here, Commas are used to separate two symbols and it is also treated as a valid special.

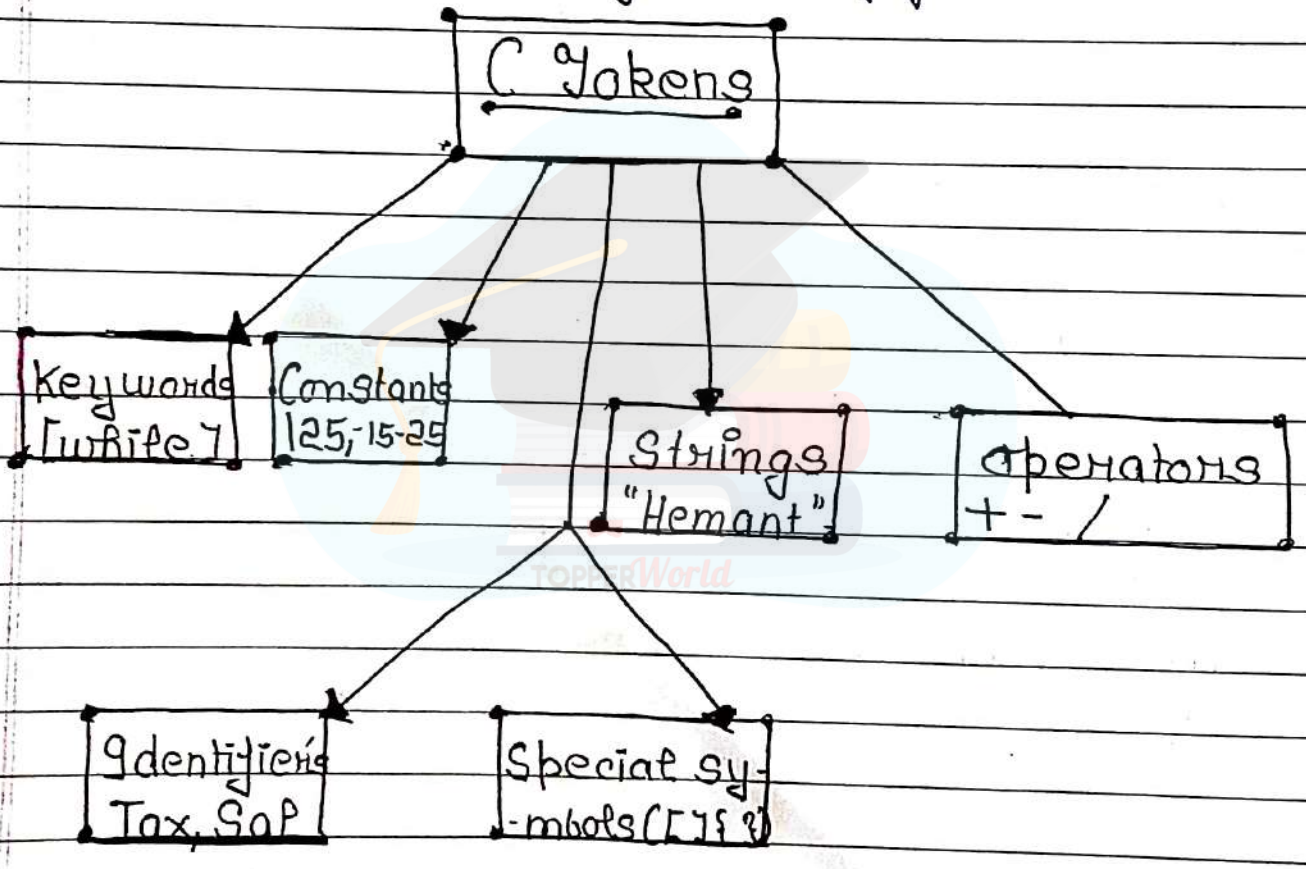
1.] **Alphabets :-** C Character Set supports alphabets from A to Z and a-z. Note that C is case sensitive language which means capital (A-Z) is different than (a-z)

2.] **Digits :-** C character set supports digits from 0 to 9. Any combination of digits from this range is valid in C.

3.] Special characters :-

In addition to alphabets and digits. C support so many special characters.

5.] C Tokens :- In C language, the smallest individual Units are called as tokens. Tokens in C are of different types. These are represented by below figure.



[Types of C Tokens]

6.] C Keywords :- Every word in a program of C language will be a keyword or an identifier. C compiler has 40 keywords. The list of these keywords are as follows

C Keywords

auto	break	case	char	const	Continue	
default	do	double	else	enum	extern	
Float	For	For	goto	if	Int	
long	near	register	return	short	signed	
static	struct	switch	typedef	union	unsigned	
void	while					

C has 32 general purpose and 8 special purpose keywords.

5.4 Rules For declaring Identifiers:-

4.4 The first character of an Identifier must be an alphabet or underscore.

2.4 It can have letters, digits and valid special characters.

3.4 C will take only first 31 characters of the Identifier.

4.4 Identifier cannot be a keyword

5.4 You Cannot Use a white space in the Identifier

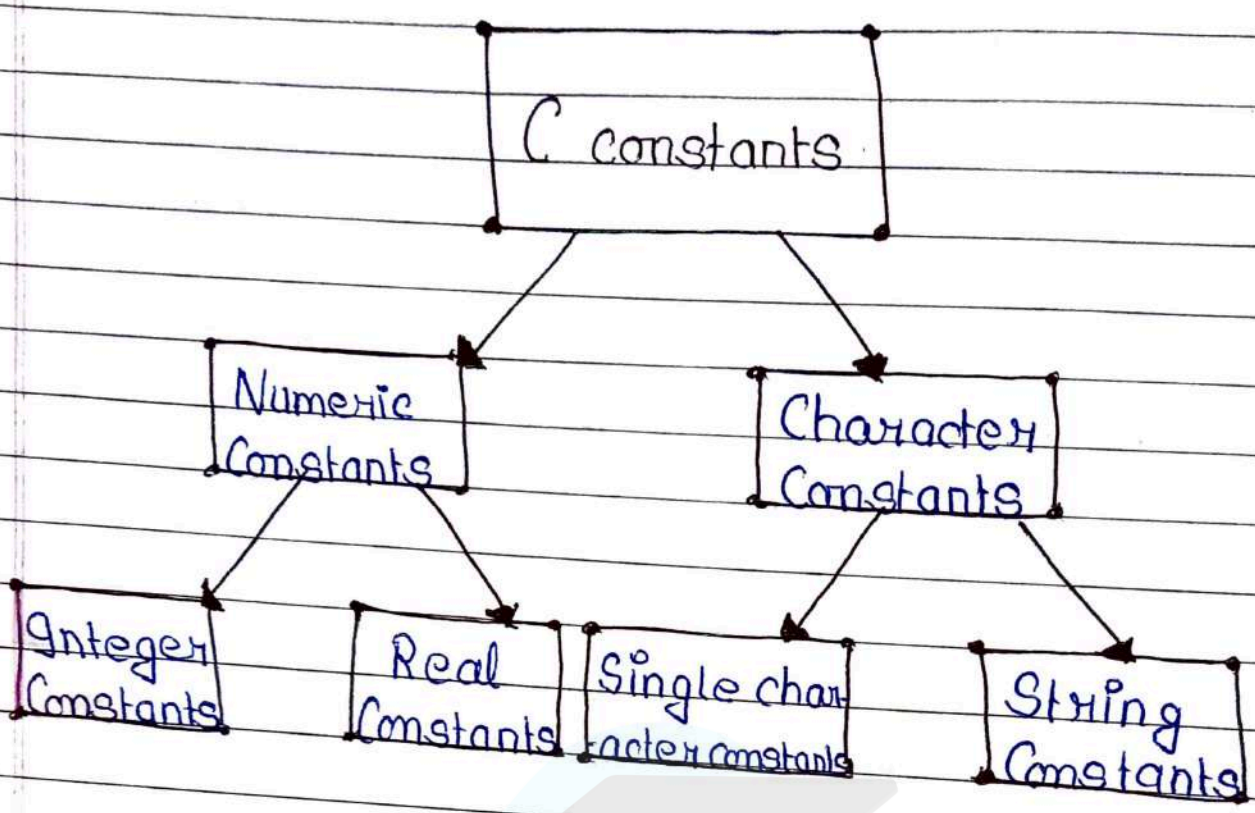
7.4 Constants:-

Constant refers to a value that do not change during the execution of a program. For Example mathematical term "Pie" is always assumed to be equal to 22/7 or 3.14.

eg $6x + 4y = 95$

In above example 6, 4, 95 values will never change

8.) Types of C Constants:-



1.) Numeric Constants:- These are the constants that are formed by combination of digits from 0 to 9. They may or may not have a decimal point in them.

i.) Integer Constants:- Integer constants are the sequence digits that are used in C programs. Decimal integer, octal Integers and hexadecimal Integers are the three types of numeric constants i.) Octal Integer constants consists of combinations of digit from 0 to 7. These are preceded by leading 0 or O. for example 025, 05245.

ii.) Hexadecimal Integer constants are represented by combinations of digits from 0 to 9 and

-racter 9 from A to F or alone - 0 to 15 Digit from 10 to 15 are represented by characters A to F respectively.

Rules For Constructing Integer constants :-

1-) An Integer Constant must have atleast one digit
2-) It must not have a decimal point.
3-) It can be either positive or negative
4-) It is assumed to be positive, if there is no sign Specified.
5-) Spaces and Commas are not allowed within it.
6-) It can have values from -32,768 to 32,768
Example : 550, -665

2) **Real Constants** :- Real constants are used to rep -resent the quantities that vary rapidly. The examples of such quantities are distances, weights, heights and prices. Real const-ants are also called as floating point Constants Real constants are further of two types :-
1-) Real Constant in fractional form
2-) Real Constant in exponential form.

Rules for Construction of Real Constants in fractional form :-

1-) It must have atleast one digit
2-) It must have a decimal point
3-) It could be either positive or negative.
4-) Positive is considered as the default sign.

5-) Commas and Spaces are not allowed in real Consta
Example : 54-564, -65-235

Real Constants in fractional form are not adequate for representing very small and very large numbers. Due to this exponential form are used - for Example 35000000000 can be written as 3.5E9 when represented in exponential form a real constant contains two parts : mantissa and the exponent

Mantissa :- Part appearing before E is called as Mantissa

Exponent :- Part appearing after E is called as exponent.

for Example 125.3E5, 125.3 is mantissa part and 5 is exponent part.

Rules for Construction of Real Constants in exponential form :-

1-) The mantissa and exponent part must be separated by character E
2-) The mantissa may be positive or negative
3-) The exponent must Contain number which must be an Integer of any sign
4-) The default sign for both mantissa and the exponent is positive
5-) It can have value from range 3.4E38 to 3.4E

Some valid real constants expressed in exponential form are +2-8e-4, 2-3e2, -0-5e-

2.] Character Constants :- In addition to numeric constants, the program also need to deal with characters and strings. Character Constants are used to express quantities like name, place or gender. They can be a single character or a group of characters or backslash character Constants-

i.] Single Character Constant :- Any character or digit within single quotes will be treated as single character constant. Some valid character constants are: 'a', 'c', '1', '2', 'x'

ii.] String character Constants :- A group of characters (constant contains a single charact.) is called as string. These constants are enclosed with double quotation marks. some valid string Constants are "Hemant", "shiva", "7898", "5+7". any group of characters or digits within double quotes will be treated as string

iii.] Backslash character constants :- In addition to above described character Constants, C Language supports some special character constants that are called "Escape sequences". Backslash character constants or

escape sequences are used with output function of C. Each of these characters has a specific purpose. They have been illustrated in following table

Constant	Meaning	Constant	Meaning
'\a'	Bell alert	'\v'	Vertical tab
'\b'	Backspace	'\''	Single quote
'\f'	Formfeed	'\"'	Double quote
'\n'	New Line	'\?'	Question mark
'\r'	carriage Return	'\\'	Backslash
'\t'	horizontal Tab	'\0'	Null

9.4 Variables :- In C programming a variable is a container (storage area) to hold data. Variable is a name, given to a temporary memory location that has been used to store any value (during program execution A variable is needed). It is clear from the name that value stored in variable can be changed at any instance of time during the program execution

example $5x+3y=77$

In above equation x and y are variables, which can have any values. 5, 3, 77 are constant.

Rules for Construction of Variables :-

The name of the variable can be any combination of single to 31 alphabets, digits or Underscore. These rules may vary with some

compilers
2:) Variable name must start with a character.
3:) Blank spaces and commas are not allowed within a variable name.
4:) No special symbol, except the underscore (_) is allowed
5:) The variable name cannot be same as any key-word.
6:) Some examples of valid variable names are: H_1, hem, salary, tabill, bro etc

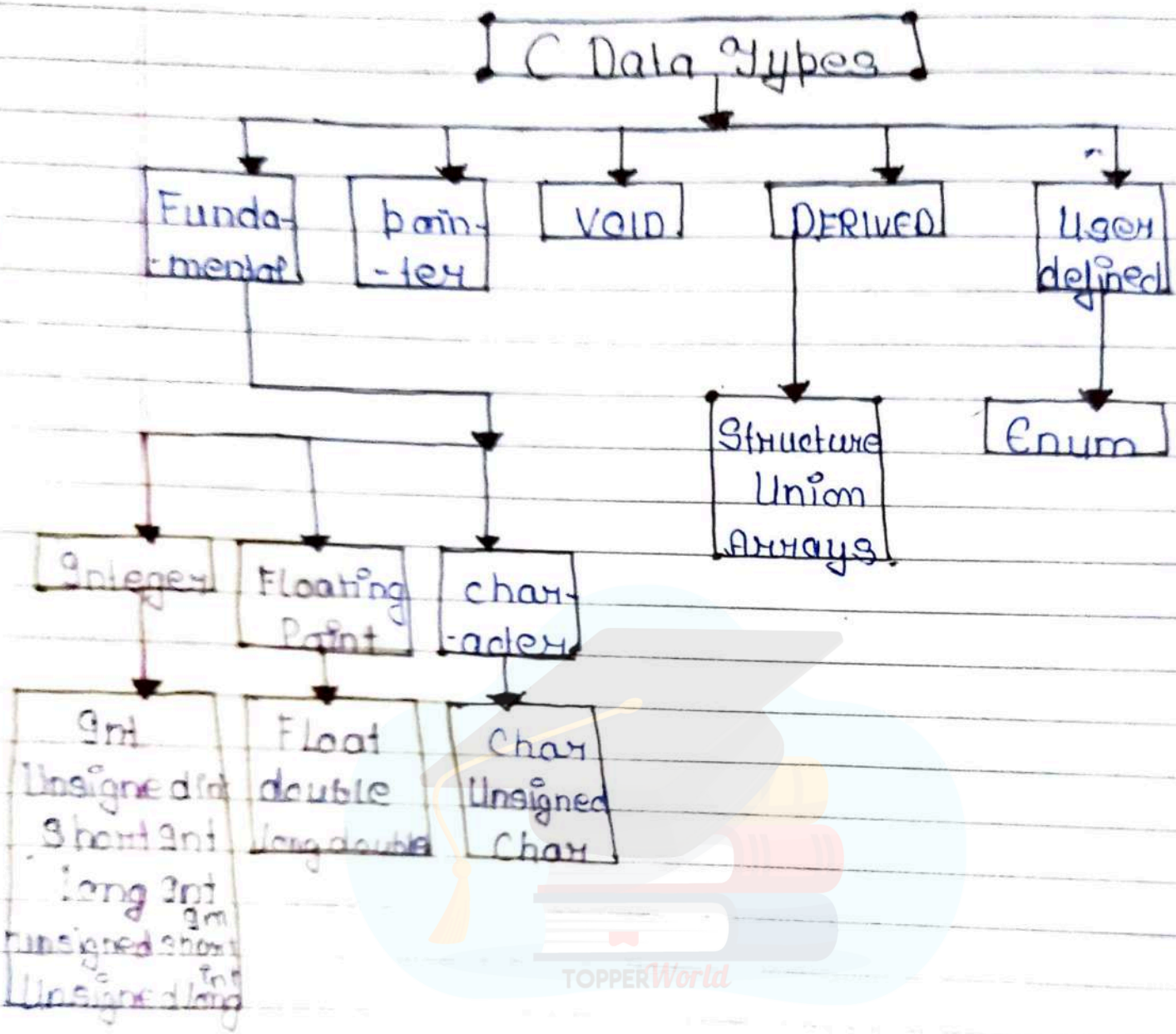
10:) Variable Declaration :- Variable is declared by using a statement that contains the data type of variable and name of variable. This statement can also contain the initial value of variable. For example following statement declares a variable basic_sal of Integer type and initialises it to 3400

```
int basic_sal = 3400;
```

11:) Datatype :- Datatype represents type of input to be carried in the variable. The data types can be classified into primary or Fundamental data types, derived data types and User defined data types. Pointers and Void data types are other supported data types by C language

1:) Primary data types :- Four fundamental data-types are supported by all C compilers. These are 1:) Integer

2) Character 3) Float 4) Double precision float



1) Integer data type:- Integer data types are used to store numeric data items. They can store whole numbers as well as numbers with fractions. The data types of this category are Integer, Short Integer, Long Integer. In addition to these types unsigned Int, unsigned short int and Unsigned long int data types also fall under this category. Following table describes all Int data types.

	Datatype	Description	Range	Bytes
1.	Int	Stores Integer (non-fractional) values both +ve and -ve	-32768 to +32,768	2
2.	Unsigned Int	Stores Integer (non fractional) values	0 to 65,535	2
3.	Short int	Stores Integer (non fractional) values both +ve and -ve	-128 to 127	1
4.	Unsigned Short Int	Stores Integer non fractional values	0 to 255	1
5.	Long int	Stores Integer values +ve and -ve	-2,147,483,648 to 2,147,483,647 (two billion one hundred forty seven million)	
6.	Unsigned long Int	Stores Integer values	0 to 4,294,967,295	4

2. Floating Point data type:- Floating Point data types are used to store numbers with fractions. Following tables describes all floating point data types.

Datatype	Description	Range	bytes
Float	used to store fractional values	$3.4^{*}10^{-38}$ to $3.4^{*}10^{38}$	4

3.) Double data types :- Float data type is not capable of storing very large numeric and floating quantities. for this double precision data type is used. following double data types described by below table.

DATATYPE	Description	Range	Bytes
double	Stores fractional values	$3.4^{*}10^{-308}$ to $3.4^{*}10^{308}$	8
long double	Stores fractional values	$3.4^{*}10^{-4932}$ to $3.4^{*}10^{4932}$	10

4.) Character data types :- character data types are used to store character data items. The values to be stored in them are enclosed in single quotation marks (''). char and Unsigned char are two character data types

Datatype	Description	Range	Bytes
char	Stores character values	-128 to 127	1
unsigned char	Stores character values	0 to 255	1

2) Derived data types :- Derived data types are also called as structured data types. The main difference between fundamental and derived data types is that a variable of derived data type can handle more than one values at a time - whereas a variable of fundamental data types can handle only one value at time. The data types in this category are arrays, strings, structure and unions.

3) User defined data types :- These are the data types that are defined by user - They contain user defined specified values. Enum keyword is used to declare data types of this type.

4) Pointer data types :- Pointer is a data type that handles the data as per its memory address for example an integer variable holds an integer value however integer pointer holds address of integer vari

5) Void data type :- Void data type represents empty or null. This datatype is used when a particular function does not return any value.

Compiled and executed

7-] To see the output Press Alt + 5 from keyboard

6-] **Errors in C Language** :- Errors are the problems or the faults that occur in the program. There are mainly five types of errors exist in C Programming.

1-] Syntax Error
2-] Run time error
3-] Linker error
4-] Logical error
5-] Semantic error

1-] **Syntax Error** :- Errors that occur when you vio-te the rules of writing C syntax are known as Syntax errors. All these errors are detected by Compiler and thus are known a Compile time error's. Most Frequent Syntax er-ors are

1 Missing Parenthesis (})
1 Printing the value of variable without declaring it.
1 Missing Semicolon like this.

```
// C program to illustrate
// "Syntax error"
#include<stdio.h>
void main()
{
int x = 10;
int y = 15;
Printf("%d", (x, y)
}
```

Error

Error : expected ';' before '}' token

Run time error : Errors which occur during Program execution (run-time) after successful compilation are called run time error's. One of the most common run time error is division by zero also known as divison error. These types of error are hard to find as the Compiler does not point to the line at which the error occurs. Example of run time error is :-

```
// C program to illustrate
// run time error
#include <stdio.h>
Void main()
{
int n=9, div=0;
// Wrong Logic
// number is divided by 0
div = n/0;
Printf("result = %d", div);
}
```

Errors will appear as :-

Warning division by zero [-wdiv-by-zero]
div = n/0;

Linker Errors : These error occurs when after compilation we link the different object files with main's object using ctrl+F9 key (RUN). These are errors generated when the

executable of the program cannot be generated this may be due to wrong function prototyping, in-correct header files. one of the most common linker error is writing Main() instead of main()
Example of linker error is :-

```
// c program to illustrate
// linker error
# include <stdio.h>
Void Main() // Here Main() should be main()
{
int a = 10;
Printf("%d", a);
}
```

Error :-
(.text+0x20); undefined reference to 'main'

4. Logical error's :- On compilation and execution of program desired output is not obtained when certain input values are given. These types of errors which provide Incorrect output but appears to be error free are called logical errors. These are one of the most common errors done by beginners of programming.

Example of logical error are :-

```
// c program to illustrate logical error
int main()
{
int i = 0;
// logical error : a semicolon after loop
```

```
For(i=0; i<8; i++);
{
printf("loop");
Continue;
}
getchar();
Return 0;
}
```

5.] Semantic Errors :- This error occurs when the statements written in the program are not meaningful to the compiler. Example of Semantic Error is

```
//C program to illustrate Semantic error
Void main()
{
int a,b,c;
a+b=c; //Semantic error
}
```

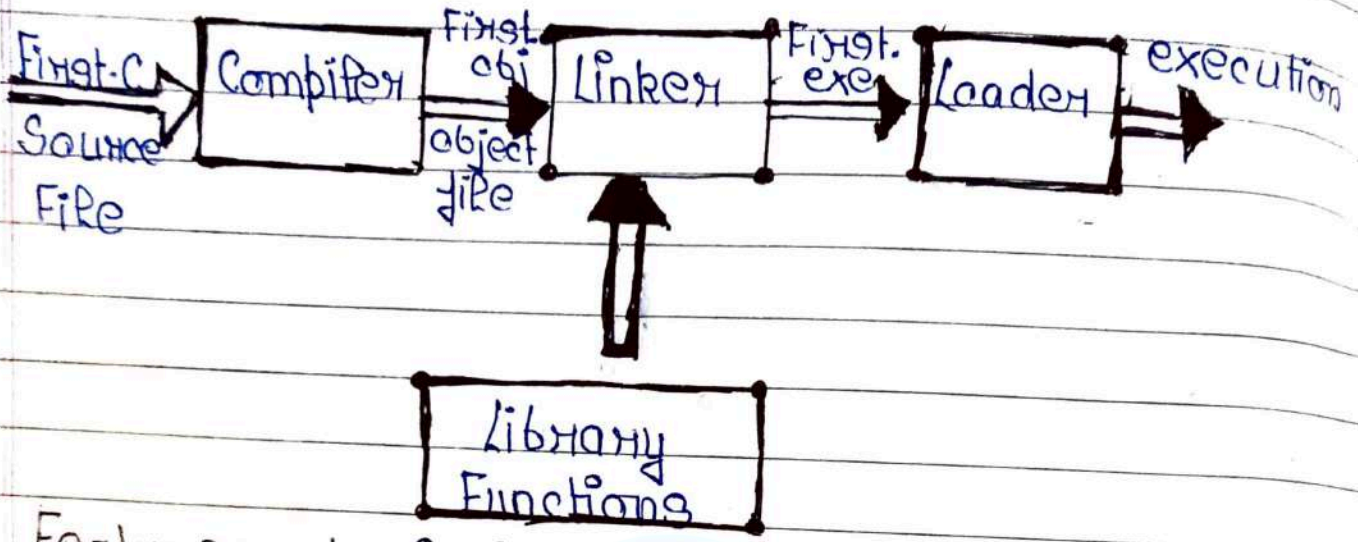
Error :-

Error : lvalue required as left operand of assignment a+b=c; // Semantic Error

17.] How does a program Executes :- whenever a C Program file is compiled and executed, the Compiler generates some files with same name as that of C Program file but with different extensions

P

Below figure shows the compilation process with the files created at each step of the compilation process.



Features of C Programming :- The main key features of C Programming are as follows :

1. Speed :- C comes with support for system programming and hence it compiles and executes with high speed when compared with other HLL languages.

2. Flexibility :- The second feature of C programming is flexibility. Flexibility means possibility of programmer to control the language

3. Modularity :- Possibility to break down large programs into small modules using sub-routine.

4. Portability :- It is a platform Independent programming language

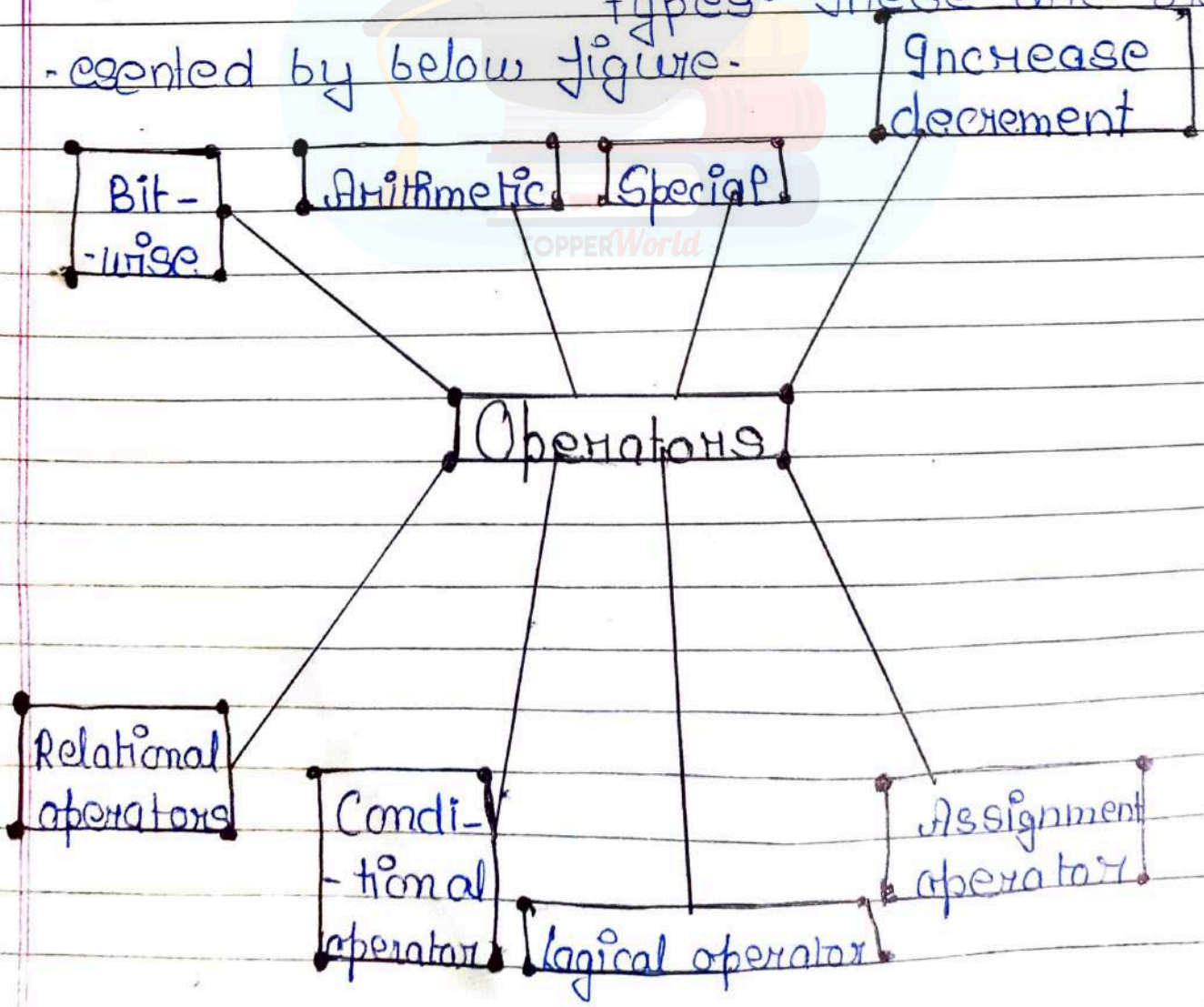
5. Extensibility :- Possibility to add new features by programmer.

Chapter - 3
Operators and Expressions.

1.] Operators :- An operator is a symbol that instructs the computer to perform a specific operation. This operation varies with the type of operator. The main use of operators is to manipulate the data and variables.

2.] Expression :- The statement containing variables and operators is called as an expression.

3.] Types of operators :- Operators are of Eight types. These are represented by below figure.



1) Arithmetic operators :- C language supports arithmetic operations on variables through various arithmetic operators. following are the operators to perform arithmetic operations.

a) Addition operator :- Addition operator "+" is used to add two or more integers or floats.

b) Subtraction operator :- Subtraction operator "-" is used to calculate the difference of two numbers or floats.

c) Multiplication operator :- Multiplication operator "*" is used to calculate the product of two or more integers or floats.

d) Division operator :- Division operator "/" is used to calculate the quotient after dividing two number. This operator has no concern with remainder, left after division.

e) Modulas operator :- The modulas division operator "%" is used to calculate the remainder left after the division of two numbers. This operator has no concern with quotient left after division

2.4 Relational operators :- Relational operators are used to compare two values Comparison is very important in a program and it acts as a base for executing certain statement in program. An expression containing one or more relational operators is called as relational expression. Relational operators are

1.4 Equal to operator (==) :- is used to compare two quantities for equality. The expression will return true if compared variables contain equal values. otherwise expression result FALSE.

2.4 less than operator (<) :- Compares the variables and results TRUE, if the value in the variable on left side of operator is less than value of variable present on the right hand side of the operator.

3.4 less than or Equal to operator :- (<=) Compares the quantities and results true if value of variable on left side of operator is less than or equal to the value of variable present on the right hand side of the operator.

4.4 Greater than (>) :- compares the variables and results true if value in a variable on left side of operator is greater than the value of variable present on the right hand side of operator.

5.4 Greater than or equal to operator :- (>=) Compares

the quantities and results true, if value in the variable on left side of operator is greater than or equal to the value of variable present on the right hand side of the operator.

6-] Not equal to (!=) Compares the quantities and results TRUE, if the quantity on left side of the operator is not equal to the quantity present on the right hand side of the operator.

3-] Logical operators :- C language is equipped with a support for logical operators to join multiple conditional expressions. These operators are used to combine two or more relational expressions to return the combined results of the relational expressions depending upon the operator used. Logical operators form logical expressions, which result either true or false. Logical operator are 1-] And 2-] OR 3-] NOT. Logical operators are used to connect relational expressions.

1-] AND operator :- Combines two or more relational expressions and results true when all the relational expressions individually return TRUE otherwise. It returns false. It is represented by '&&' symbol. For example the the expression "(x<4) && (x<z)" will result TRUE if (x<4) is true and (x<z) is also true else, it will return false.

2-] OR operator :- Combines two or more relational expressions and results true when

either one or more of relational expressions, individually return true. It returns false when all individual relational expressions result into false It is represented by '||' symbol

iii. NOT operator:- is used to reverse the result of relational expression. It just negates the obtained result. It is represented by exclamation symbol '!' for example expression "!(x>4)" will result false. If x is greater than 4 and will result true, if x is less than 4. So result is exactly opposite to that of (x<4)

4. Assignment operators:- "=" is called Simple Assignment operator of C language. It is used to assign values to variables in C. In C language, the variable on the left hand side of the assignment operator receives the value of variable or expression present on the right hand side of assignment operator. For example in the expression x=5 5 will be stored in variable "x" In expressions "a=z+1" or "a=c+10", result of expression will be stored in "a".

5. Increment and Decrement operators:- In addition to standard arithmetic operators, C also provides special operators for incrementing and decrementing the value stored in variable ++ is called increment operator and -- is called decrem

-ent operator. If you use ++ with variable, its value will be incremented by 1 and if you use -- the value of variable will be decremented by 1.

i) Prefix Increment or Decrement operators: If the operators are used before the variable name, they are called as Prefix increment or decrement operator. In case of increment decrement operator the value of variable is first incremented and then used in statement. These operator's are represented as

a) ++A: Prefix increment the value of A that means if current value of A is 5 it will become 6 the equivalent simple statement is A = A + 1.

b) --A: Prefix decrement the value of A that means if current value of A is 5 it will become 4. the equivalent simple statement is A = A - 1.

ii) Postfix Increment or Decrement operators: If the operators are used after the variable name, they are called as Postfix increment or decrement operator. In case of this operator, the value of the variable is first used in current statement and then incremented or decremented. the Postfix increment or decoement operator

are represented as

A++ : Postfix increment the value of A. That means, if current value of A is 5 it will become 6. The equivalent simple statement is A = A + 1.

A-- : Postfix decrement the value of A. That means if current value of A is 5, it will become 4. The equivalent simple statement is A = A - 1

Conditional operator : Conditional operator is condensed form of if, else, else statement. Conditional operator is used to assign a value to a variable depending on the result of a conditional expression. The syntax of Conditional or ternary operator is stated as follows. It operates on three values that's why it is called Ternary operator.

Expression 1 ? Expression 2 : expression

This operator can help programmers in performing simple conditional statements.

Bit wise operators : C allows you to manipulate data items at bit level. Most of these operators are used in advanced system programming or some special tasks like encryption and data compression. C supports 6 bit wise operators that change the bits of data items following is the list of Bit wise operators and their description. manipulate the bits. This operator is used in C to perform bit-level operations.

operator	Description
>>	Right shift
<<	Left shift

Page

~	1's Complement
&	Bitwise AND
|	Bitwise OR
^ [XOR]	Bitwise Exclusive OR

Hello ... I'm my ... when do I stop?

1. Bitwise AND operator [&] The output of bit-wise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 [In Binary]
25 = 00011001 [In Binary]

Bit operation of 12 and 25

```
  00001100
& 00011001
  00001000   [In decimal = 8]
```

Example Program :-

```
#include <stdio.h>
Void main( )
{
Int a = 12, b = 25;
Printf ("Output = %d", a & b);
}
```

Output will display as

Output = 8

2.) Bitwise OR operator :- [|]

The Output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C programming bitwise OR operator is denoted by |.

Let us suppose that bitwise OR operation of two integers are 12, 25

12 = 00001100 [In Binary]
25 = 00011001 [In Binary]

Bitwise OR operation of 12 and 25

```
  00001100
| 00011001
  00011101
```

Example :-

```
#include<stdio.h>
Void main()
{
Int a = 12, b = 25;
printf ("Output = %d", a|b);
}
```

Output will display as
Output -- 29

3-] Bitwise XOR :-

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

Let us suppose that Bitwise XOR operations of two integers are 12 and 15.

12 = 0000 1100 [In Binary]
15 = 0000 1101 [In Binary]

Bitwise XOR operation of 12 and 25

```
    0000 1100
^   0000 1101   = 21 [In decimal]
    0000 0001
```

Example :-

```
#include <stdio.h>
void main()
{
int a = 12, b = 25;
Printf("Output = %d", a ^ b);
}
```

Output will display as

Output = 21

4-] Bitwise Complement operator is an unary operator (works on only one operand). It changes the

and 0 to 1. It is denoted by ~.

Let us suppose that Bitwise Complement operation of Integer 35

35 = 00100011 [In Binary]
Bitwise Complement Operation of 35
~ 00100011
11011100 = 220 [In decimal]

Twist in bitwise Complement operator in C Programming:
The bitwise Complement of 35 (-35) is -36 Instead of 220 but why for any Integer n, bitwise Complement of n will be (-n+1) To understand this, you should have the knowledge of 2's Complement

5. 2's Complement:-
Two's Complement is an operation on binary numbers. The 2's Complement of number is equal to Complement of that number plus 1 For Example

Decimal	Binary	2's Complement
0	00000000	-(11111111+1) = -00000000 = -0(decimal)
1	00000001	-(11111110+1) = -11111111 = -256 decimal
12	00001100	-(11110011+1) = -11110100 = -244
220	11011100	-(00100011+1) = -00100100 = -36

The bitwise Complement of 35 is 220 (in decimal) The 2's Complement of 220 is -36 Hence the output is -36 instead of 220

Example:-

```
#include <stdio.h>
void main()
{
Printf("\n output= %d", ~35);
Printf("\n output= %d", ~12);
```

Output:- Output = -36
Output = 11

6) Right Shift operator:- Shifts all bits towards right by certain number of specified bits. It is denoted by >>.

212 = 11010100 [In Binary]
212>>2 = 00110101 [In Binary] [Right Shift by two bits]
212>>7 = 00000001 [In binary]
212>>8 = 00000000
212>0 = 11010100 [No shift]

7) Left Shift operator:- Shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by Left Shift operator are filled with 0. The Symbol of left shift operator is <<

212 = 11010100 [In binary]
212 << 1 = 11010100 (In binary) (left Shift by one)
212 << 0 = 11010100 (shift by 0)
212 << 4 = 110101000000 (In Binary) = 3392 (In decimal)

Example :-

```
# Include <stdio.h>
Void main()
{
 Int num = 212 ;
 For (i = 0 ; i <= 2 ; ++i)
 Printf (" Right Shift by %d : %d \n", i, num >> 1);
  Printf ("\n");
For (i = 0 ; i <= 2 ; ++i)
Printf (" Left Shift by %d : %d \n", i, num << 1);
}
```

Special operators :-

1. Comma Operator :- Comma operator can be used to link the related Expressions together

Example :-

Comma operator in Variables

```
uint a = 3, b = 2, c = 3;
```

2.) Type operator :- Type operators a to the specified type.

Example :- type

```
Float s = 12.5; int a;
a = (int) s;
```

now a value is 12

3.) Address or Reference operator :- ("&") The address operator noted by ampersand ("&"). It is also a unary operator in C that uses for assign address of the Variables. It returns the pointer address of the Variable. This is called referencing operator.

3.) Dereference operator ("*") :- The Pointer operator is noted by asterisk ("*") is also a Unary operator in C that uses for pointer variables. Th is also is called "dereferencing the pointer because it operates on pointer variable and returns l value equivalent to value at pointer address.

Syntax :-

```
data-type *pt;
```

4.) Double pointer operator :- Double pointer is that double pointer points to another pointer Variable address.

Syntax :-

```
data_type **pt;
```

5-) Size of operator :- Size of returns the size of variables or data type

Syntax :-

Size of (data-type)

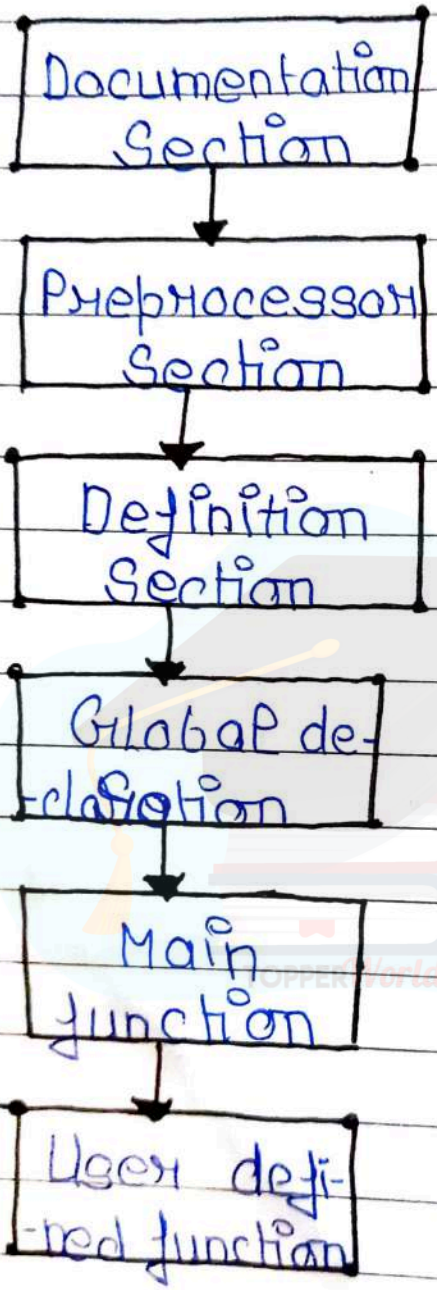
Example of Size of and & operators

```
#Include <Stdio.h>
Void main()
{
Int a = 20;
char b = 'B';
double c = 1734494.249324;
Printf("Size of a is : %d", size of (a));
Printf("Size of b is : %d", Size of (b));
Printf("Size of c is : %d", Size of (c));
Printf(" Memory address of a : %d \n", &a);
}
```

Output :-

```
Size of a is = 4
Size of b is : 1
 Size of c is : 8
Memory address of a = 1684270284
```

Structure of C :- C has different Sections. The Sections of c program are listed below:-



1. **Documentation Section** :- It includes the statement specified at the beginning of a program. Such as prog-

-ram's name, date, description and title. It is represent as

//name of a program

or

/* overview of Code */

Both methods work as the document section in a program. It provides an over-view of the program. Anything written inside will be considered a part of the documentation section and will not interfere with specified code

2. Prepprocessor Section :- The prepprocessor Sec-tion contains all the header files used in a program. It informs the system to link the header files to the system libraries. It is given by

#include<stdio.h>

include<conio.h>

The # include statement includes the specific file as a part of a function at the time of the Compilation. Thus the contents of the included file are Compiled along with the function being Compiled. The #include <stdio.h> Consists of the Contents of the standard input output files. which Contains the defination of stdin, stdout and stderr. Whenever the definations stdin stdout are used in function the statement #include<stdio.h> need to be used. There are various header files available for different purpo-

ses - for example #include <math.h>. It is used for mathematic functions in a program.

3. Define Section:- The define section comprises of different constants declared using the define keyword. It is given by

#define a = 2

4. Global declaration:- The global section comprises of all the global declarations in the program. It is given by

```
Float num = 2.54;
int a = 5;
char ch = 'z';
```

The size of the above global variable is listed as follows:-

char = 1 byte
Float = 4 bytes
Int = 4 bytes

We can also declare User defined functions in global variable section.

5. Main function:- main() is the first function to be executed by the Computer. It is necessary for a code to include the main(). It is like any other function available in C library. Parenthesis () are used for passing parameters (if any) to a function.

The main function is declared as [

```
main()
```

we can also use int or void main with the main(). The Void main() specifies that the program will not return any value. The int main() specifies that the program can return Integer type data.

```
int main()
```

or

```
Void main()
```

Main function is further categorized into local declarations, statements, and expressions.

Local declarations: The Variable that is declared inside a given function or block refers to as local declarations:-

```
main()
{
int i=2;
i++;
}
```

Statements:-

The Statements refers to if, else, while, do for etc used in program within main function.

Expressions:-

An expression is a type of formula where operands are linked with each other by the use of operators. It is given by

```
a-b;
a+b;
```

5.4 User defined functions :- specified the functions specified as per the requirements of the user. for Example Color( ), Sum( ), division( ) etc. The program follows the same section as listed above

Return function is generally the last section of code, but it is not necessary to include. It is used when we want to return a value. The return function returns a value when return type other than the void is specified with the function. Return type ends the execution of function. It further returns control to specified calling function It is given by

return;
or
return expression;

For Example
return 0;

Sample Program Output :- Programming in C is easy and also programming in C++

Example :-

```
#include <stdio.h>
Void main ( )
{
Printf ("Hello ..... \n. of my \n when do a
Stop ?. \n");
}
```

Operator Precedence :-

Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence Certain operators have higher precedence than others - for example the multiplication operator has higher precedence than addition operators

for example $x = 7 + 3 * 2$;; x is assigned 13 not 20 because * operator has higher precedence than + operators. So it first multiplied with 3*2 and then adds into 7.

Category	operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & size of	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	<< = >> =	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example :-

```
#include<stdio.h>
main()
{
int a=20, b=10, c=15, d=5, e;

e= (a+b) * c /d;
Printf("Value of (a+b)*c/d is %d \n",e);

e =(a+b * c) /d;
Printf(" Value of ((a+b) *c) /d is= %d \n" e);

e =(a+b)*(c/d);
Printf("Value of (a+b) *(c/d) is = %d\n",e);

e = a+(b*c) /d;
Printf("Value of a+(b*c)/d is=%d\n"e);
}
```

Output will be :-

```
Value of (a+b) *c /d is = 90
Value of ((a+b) *c /d is= 90
Value of (a+b)*(c /d) is =90
Value of a+(b*c)/d is =50
```

Calculate the following Expression In C Language precendence

1. 100 + 200/10 - 3*10
   100 + 200/10 - 30
   100 + 20 - 30
   = 120 - 30

2. Program in C of this expression in C is :-

```
#Include <stdio.h>
#Include <conio.h>
Void main()
{
Int a = 100, b=200 c=10 d=3 e=10;
Heir;
Heir= a + b/c - d*e
Prinf("\n output is = %d", Heir);
getch();
}
```

3. 250/50 + 300 * 2 + 450%70 - 35
   250/50 + 600 + 450%70 - 35
   = 5 + 600 + 450%70 - 35
   = 5 + 600 + 30 - 35
   = 605 - 5
   = 600

Program for this Expression is :=

```
#Include<stdio.h>
Void main()
{
```

```
int a=250, b=50, c=3000, d=2, e=3000, d=2, e=450,
f=70, g=35, pro;
clrscr();
pro = a/b + c*d + e%f - g;
printf("\n output is = %d", pro);
getch();
}
```

3.] 5+2*10/7+13%5+19/2-5*2-1

=5+20/7+13%5+19/2-10-1

=5+2+3+9-11

=7+3-2

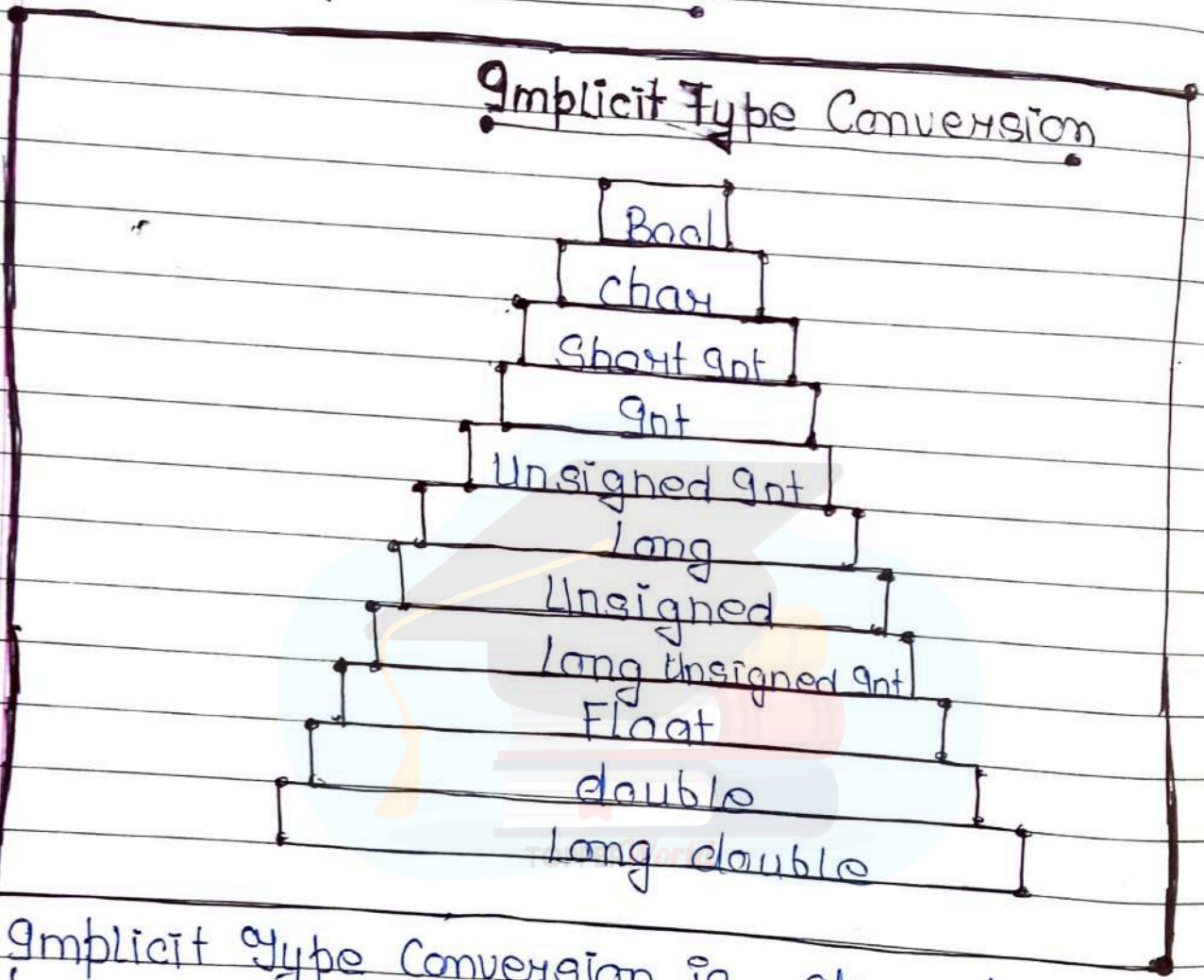
=10-2=8

Program for expression is :-

```
#include<stdio.h>
Void main()
{
int a=5, b=2, c=10, d=7, e=13, f=5, g=19, h=2, i=5, j=2,
k=1, hier;
clrscr();
hier = a+b*c/d+e%f+g/h-i*j-k;
printf("\n output of Expression is = %d", hier);
getch();
}
```

Type Conversion in C :- A type cast is basically a conversion from one type to another. There are two types of type conversion.

1. Implicit Type Conversion :-



Implicit type conversion is also known as 'automatic type conversion'. Implicit type Conversion is done by the compiler on its own, without any external trigger from the user. This conversion generally takes place when in an expression more than one data type is present. In such condition type conversion takes place to avoid loss of data. All the data types of the variable are upgraded to the data type of the

Variable with largest data type.

Bool → char → short int → int → unsigned int → long → unsigned long long → float → double → long double

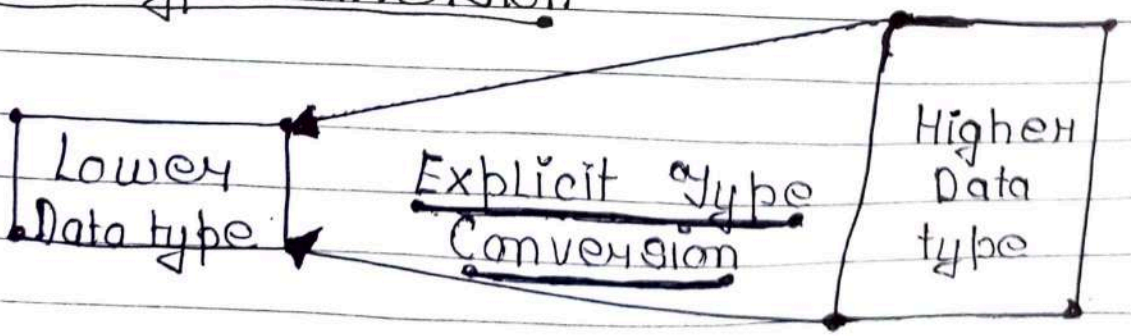
It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned.) and overflow can occur (when long long is implicitly converted to float).

Example of implicit type Conversion

```
// Example of implicit type Conversion
#include <stdio.h>
Void main ()
{
Int x = 10;
Char y = 'a';
x = x+y;
Float z = x+1.0
Printf ("x =%d, Z = %f", x, z);
}
```

Output:
X = 107, Z = 108.000000

2.4 Explicit type Conversion



This process is also called type Casting and it is user defined - Here the user can type cast the result to make it of particular data type

The syntax in C is

(type) expression

type indicated the data type to which the final result is converted

```
#include<stdio.h>
void main()
{
double x = 1.2;
int sum = (int)x + 1;
printf("sum = %d", sum);
}
```

Output:-

Sum = 2

Advantages of type Conversion :-

1. This is done to take advantage of certain features of type hierarchies or type representations.

2. It helps us to compute expressions containing variables of different data types.

Chapter - 4
Input and Output operations

4.1 Console Input/output in C Programming :-
In order to keep C programming language compact Dennis Ritchie removed anything related to input or output from the definition of the language. Therefore C has no provisions for input and output of data from input and output devices. In order to solve this little discrepancy, C developers developed several standard input and output functions and placed them in C libraries. All these libraries are accessible by all C compilers

Types of Input/output functions :- A lot of input/output functions have been defined in standard libraries. These can be classified as follows

1. Console I/O functions :- These functions allows us to receive input from input devices like keyboard and provide output to output devices like visual display unit.

2. File I/O functions :- These functions allow us to access the hard disk or floppy disk to perform input and output.

Console I/O functions :- A Console Comprises the VDS and the keyboard the Console input and output functions can be classified into two categories

4.4 Formatted Console Input functions :-

4.4 Scanf() :- Scanf() is the formatted Console input function which reads formatted input from Stdin(Standard input). It can read any integer, float, character string etc data from the user. The syntax of using Scanf() is as follows:

Syntax

Scanf ("Format Specifier", arguments address);

Example :-

Scanf ("%d", &sum);

In this example %d is format specifier for an integer. & num indicates the address of num where value is to be stored under the variable name 'num'.

Disadvantages :-

One disadvantage of scanf() when taking string input is that it will ignore the string that has been entered after a blank space. Thus, Scanf() does not take multi-word string inputs. To take multi word string input we should use gets() method.

2.) Printf() :- Printf() is the formatted Console output function which prints the formatted output to the stdout (Standard output) It can display integers, floating point values, characters, string etc as indicated by the User. The Syntax of using printf is as follows

```
printf("text");
```

This shall simply print "text" on output screen. In order to print any variable value along with text we have the following format

```
printf("text <format Specifiers>", variable);
```

This shall print the value of the variable in format specified by format Specifier
For example, printf("marks: %d", marks);
This has the format specifier as %d which stand for integer data and it will print the marks in integer format.

Limitation :- For Floating point values the format Specifier is %f and it will print 6 decimal places after the floating point. For example, 3.6 will be displayed as 3.600000. In order to limit the decimal places after floating point, the format specifier can be written as %0.3f for 3 decimal places after floating point

2.) Unformatted Console I/O functions :- The Unformatted Console input/output functions deal with a single character or a string of characters. Let us see how all

these Unformatted functions work

1. getch() :- getch() function is also an unformatted input function supported by C Language. The function reads a character from the keyboard and does not echoes it to the screen. It is present in the header file "Conio.h". This function returns the character that was typed last or was typed most recently.

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main()
 {
Printf("Your name is Shubham");
getch()
 }
```

2. getche() :- getche() is an unformatted input function supported by C language. It is an unbuffered input dev function. This function waits for user to press a key from the keyboard and instantly transfers the value into variable. This function is same as getch(). The only difference is that it displays the most recently used character. It is also present in <conio.h>

Its difference from getchar() function is that it does not requires pressing of enter key after typing the character.

Syntax :-

Variable Name = getche();

Example: char rec;
rec = getche();

On executing above statement, the key pressed by user will be assigned to variable "rec".

3.4 getchar() :- getchar is an Unformatted Input function supported by C Language. This is buffered Input output function. This function is used to get or read a single character from the standard input device. The definition of this function is stored in header file "stdio.h" getchar() is a macro that works in a similar manner as getch() and also displays(echoes) the character but it needs the user to press the enter key after the character.

Syntax :-

Variable Name = getchar();
[Here variable name should be valid variable name in C, conforming to the variable naming conventions. When you execute the program, the the function waits for the user to press a key]

Example:-

```
char rec;
rec = getchar();
```

On executing above statement, the key pressed by user will be assigned to the variable "rec".

4. putchar() :- Putchar() is a single character output function. It just transmits a single character to Standard Output device (monitor). The function takes the name of variable as argument and displays the character stored in variable on screen. The definition of this function is stored in header file "stdio.h".

Syntax :-

```
putchar(variable name);
```

Example:-

```
char c;
c = 'A';
putchar(c);
```

5. gets() :- The gets() function is used to input strings from the keyboard. The advantage of this function is that it can store blank spaces as a part of string in this way, it is better than scanf() which fails to do the same. It takes

a string input (single word or multi-word) from user - it terminates when enter key is pressed

Syntax :-

gets(string name);

5. puts :- The puts() function is used to print strings on the standard display device the advantage using the function is that it automatically puts a new line character in front of the string being displayed so output always appears in new line. It is used to print string to a console.

Syntax :-

puts(string name);

Format Specifier for I/O Here's a list of Commonly used C data types and their format specifier

	Data type	Format Specifier
1.	int	%d, %i
2.	char	%c
3.	float	%f
4.	double	%lf
5.	short int	%hd
6.	unsigned int	%u

Long int	%ld
Long Unsigned int	%lu
Long long int	%llu
Unsigned char	%c

Program to Illustrate Putchar() function :-

```
// Program to demonstrate the use of Putchar()
#include <stdio.h>
main()
{
char ch;
Printf("\n Enter your Single character");
ch = getchar();
printf("\n You entered the character ->");
putchar(ch);
}
```

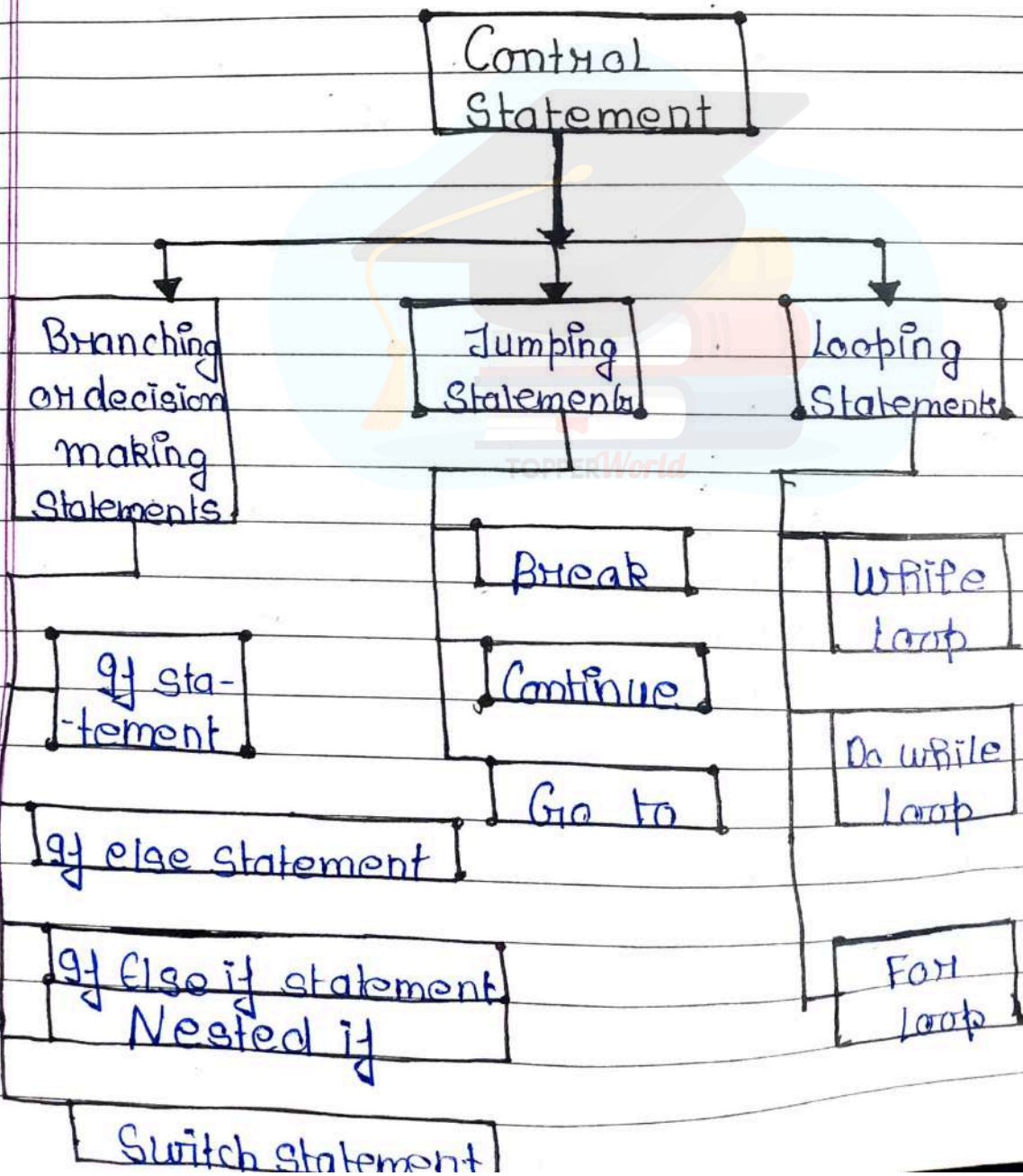
Program in C to demonstrate the use of getche();, gets, puts in C Language :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
char ch[10];
clrscr();
Printf("Enter your name");
gets(ch);
```

```
printf("\n Your name is :");
puts(ch);
getch();
}
```

Chapter - 5
Control Statements

Control Statements :- Control statements are the statements which are used to control the flow of execution of the instructions written in a program. Simply the instructions are executed sequentially. This type of control structure is known as sequence control structure.



1. Branching :- Branching is a procedure of executing a block of statements on the basis of the result of Condition. This is done through decision making statement.

2. Jumping :- Jumping is a procedure of transferring execution control to a particular point in program. This is done through jumping statements.

3. Looping :- Looping is a process of repeating a set of statements until a condition hold true. Three types of looping statements are used to accomplish this task.

Decision making Statements :- or Selection Constructs are used for Accomplishing branching in C programs. These statements select one of the two or more execution paths, depending on the result of Condition. This means, if the Condition is true, one set of statement is executed and if the Condition is false, other set of statements is executed. C provides following decision making statements

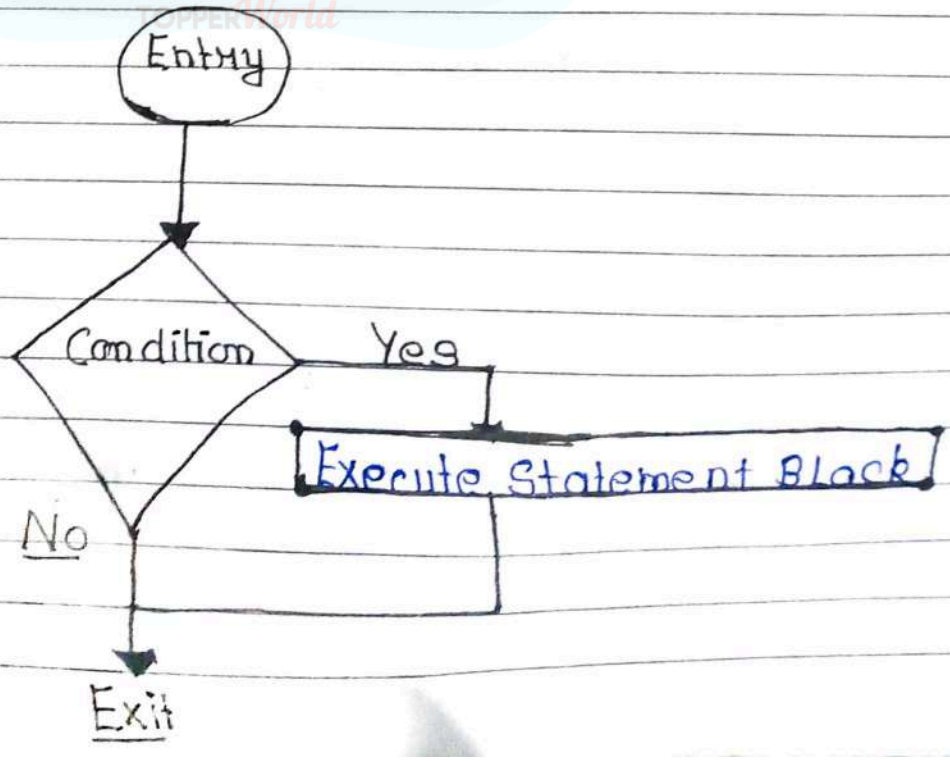
1. if statement
2. if else Statement
3. if else if statement
4. Nested if
5. Switch statement

4.1 If Statement :- If statement is a Conditional or Selection Construct that is used to execute a set of statements, if Certain Condition is true
The General Syntax of if Statement is :-

Syntax :-

```
if (condition)
{
Statement 1;
Statement 2;
- - - - - - -
}
```

Statement 1 and 2 will be executed only if the Supplied Condition results to true. otherwise the whole block will be skipped and the execution will continue after if block. Below Flowchart illustrates the use of Simple if statement



Example :-
To Check whether a student is pass in the examination

```
// Program to check whether a student is pass
# include <stdio.h>
# include <conio.h>
Void main()
 {
int marks;
clrscr();
Printf("\n Enter your marks in exam:");
Scanf(" %d", &marks);
if(marks >=35)
{
 printf("You are pass");
 printf("Congratulation");
}
getch();
}
```

Illustration :- Program inputs value of marks and stores in variable marks and displays the message of Pass and Congratulations, if the entered marks are greater than or equal to 35.

[If there is only one statement under if block then, there is no need to put Curly braces {} they are required only when the number of statements is more than one]

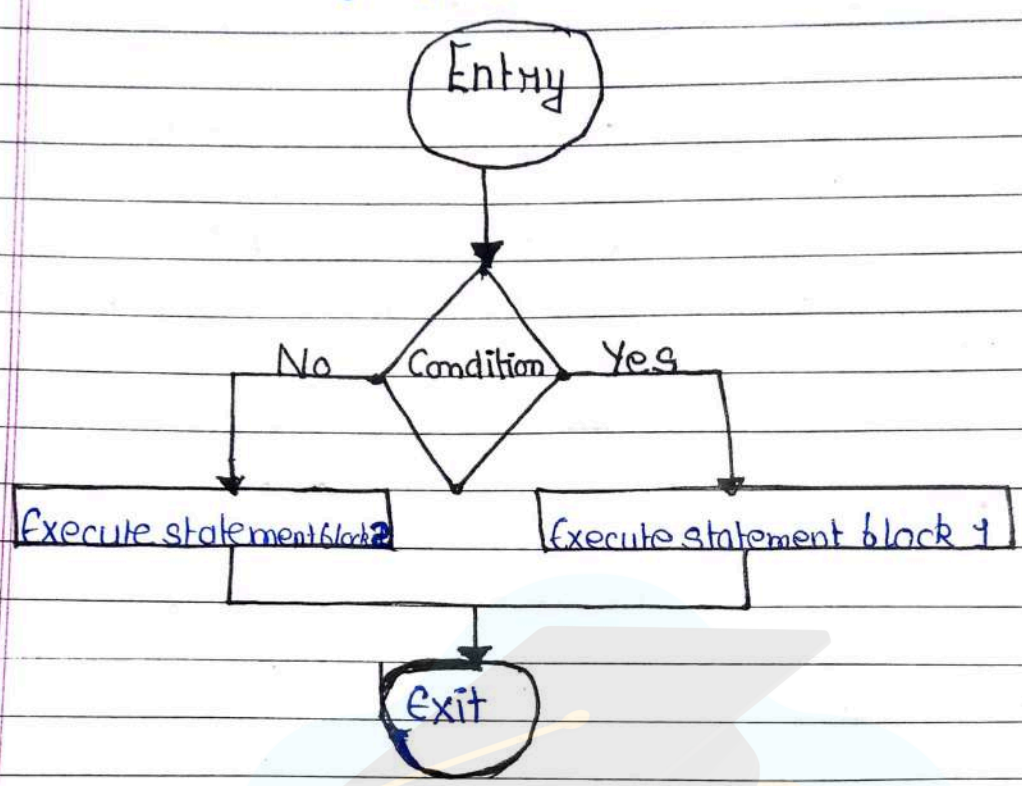
24 If Else Statement :- If else Construct is also a Conditional Construct of c language. It is used to execute a set of statements, if the condition is true and some other set of statements, if the condition is false. If else construct is an extension of if statement, it includes a condition a statement block to be executed when condition is true and a statement block to be executed, if the condition is false

Syntax :-

```
if (condition)
{
Statement block 1
- - - - - -
}
else
{
Statement block 2
- - - - -
}
```

If the condition is true then statements in the first block will be executed and if the condition is false, then the statements of the second block will be executed. if else statement is used in the situations when the conditional statement can only have two possible results, one execution path is followed when the first result is obtained and other execution path is followed.

When the second result is obtained. below Flowchart of if else logic program illustrates the working of if else Construct.



Flowchart For if else logic

Example Program:

Write a program to input a number from User and print whether the number is even or odd.

```
//Program to check whether a number is even or odd.
#include <stdio.h>
#include<conio.h>
Void main()
{
int num;
printf("\n Enter a number");
Scanf("%d", &num);
```

```
if(num%2 == 0)
{
 printf("\n You entered even number");
}
else
{
Printf("\n You entered odd number");
}
}
```

Illustration :-

Program inputs a number and stores it in a variable "num". If the value of num is further divisible by 2, the first block of statements is executed, otherwise the second block executed

3. Nested if statements :- When an if statement is written inside the body of other if statement. It is called nested if statement. In case of nested if statements, the inner if statement is executed only if the outer evaluates to true. Below Program illustrates the use of nested if statement

```
//Program to check whether a number is divisible
 by 2 and 3
#include <stdio.h>
#include<conio.h>
 Void main()
 {
  int num;
```

```
printf("\n enter a number");
scanf("%d", &num);
if(num%2==0)
{
    printf("\n entered number is divisible by 2");
    if(num%3==0)
    {
        printf("\n entered no is divisible by 3");
    }
}
getch();
}
```

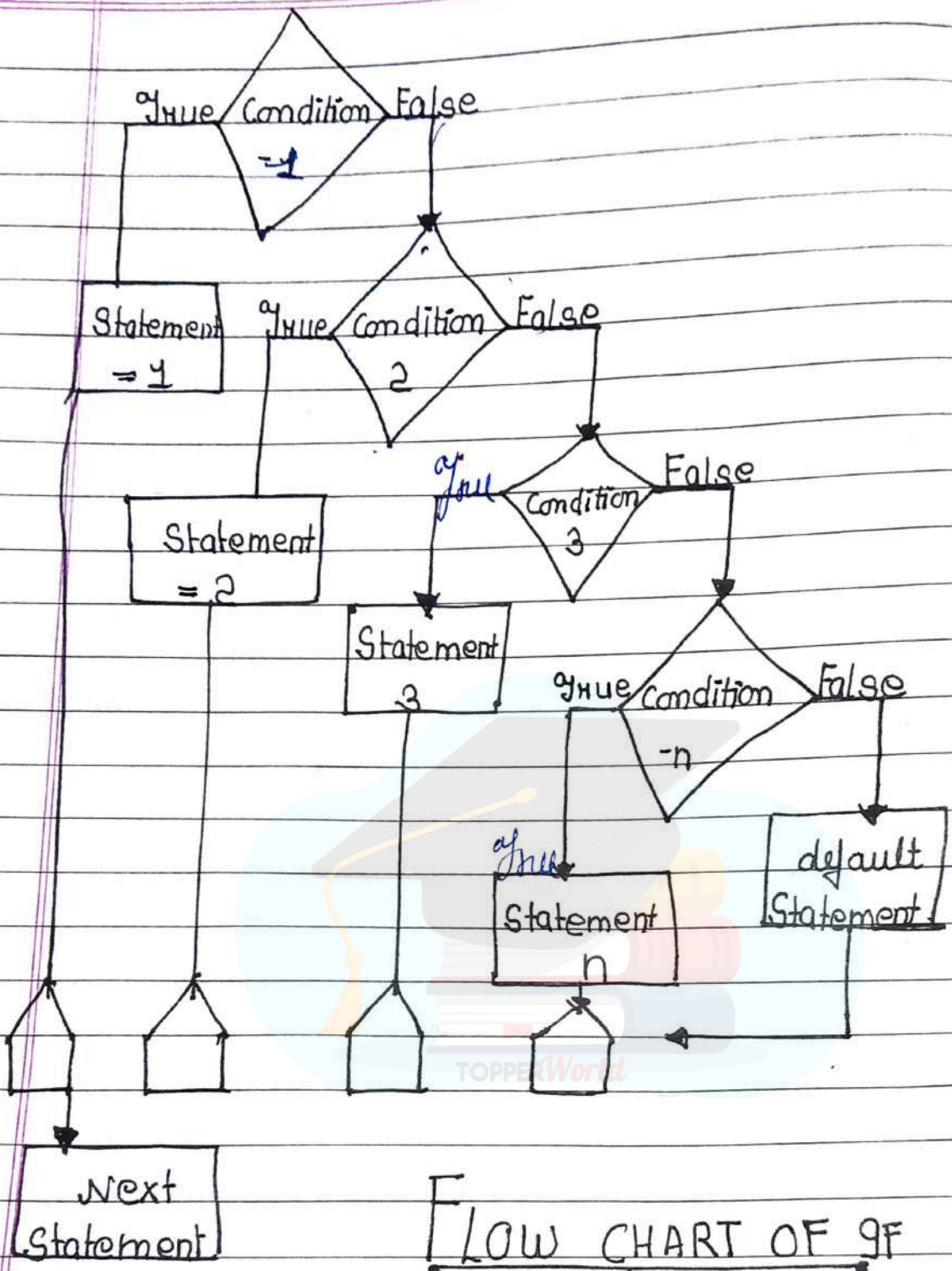
Illustration:-
The program inputs a number and stores it in a variable "num". If the number is completely divisible by 2, a message is printed and then nested if statement is checked. If the number is also divisible by 3, another message is printed.

4. If else if statement :- If else construct works for situations where there are two possible execution paths for a particular condition. In practical problems, there exist multiple execution paths for a condition. For such kind of problems, C has a support for if else if construct. This construct works with the conditions having multiple possible outcomes, If else if construct is used when the condition has multiple possible outcomes.

Syntax :-

```
if (condition 1)
{
Statement block 1
-- ---
}
else if (condition 2)
{
statement block 2
-- ---
}
else if (condition 3)
{
Statement block 3
-- ---
}
```

While executing if else if construct, c compiler checks the conditions from top to bottom. If first condition is true then statement block 1 would be executed and rest of conditions will not be checked. That means all steps till the end will be skipped and no other condition will be check. If condition 1 is false, then only Condition 2 will checked. If condition 2 is true then statement block 2 will be executed and rest all will be skipped. If condition 2 is also false, then only Condition 3 will be checked and statement block will be executed below Flowchart illustrates the implementation of if else if construct



FLOW CHART OF IF ELSE IF Logic

Program to demonstrate if else if Construct:-

```
# include <stdio-h>
# include < Conio-h>
Void main ( )
{
```

```
int mks;
printf("\n Enter your marks");
Scanf("%d",&mks);
If(mks<35)
{
printf("\n I am sorry! You are FAIL");
}
else if((mks>=35)&&(mks<45))
{
printf("\n You have got c grade");
}
else if((mks>=45)&&(mks<60))
{
Pafnf("\n You have got B grade");
}
else if (mks>60)
{
printf("\n You have A grade");
}
getch();
}
```

Initialize

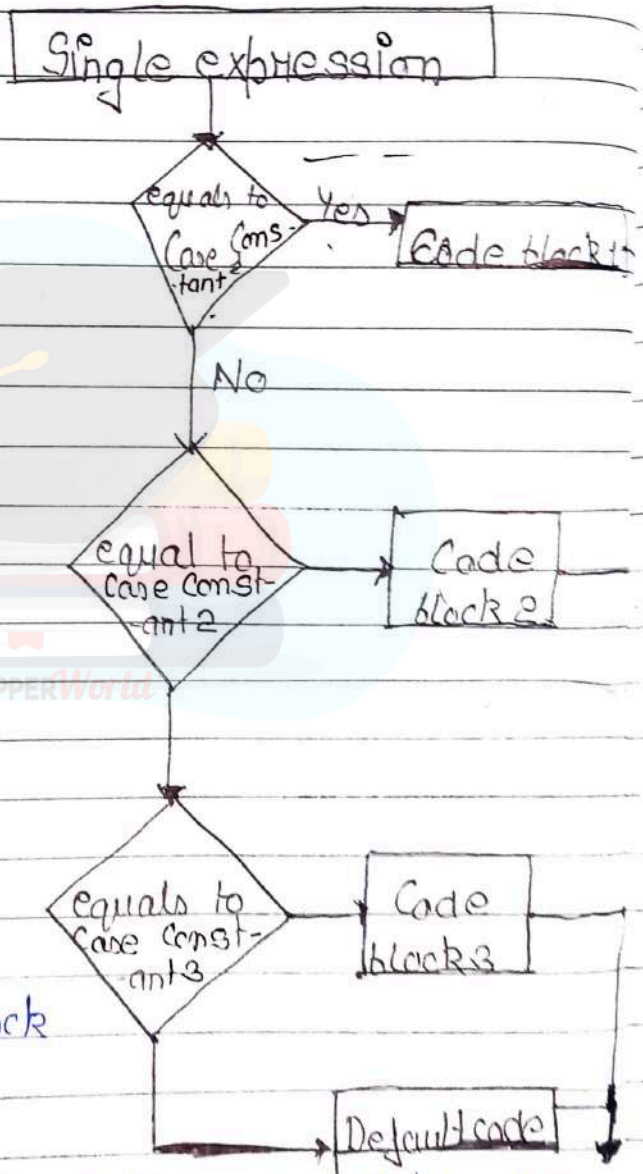
Output:-

Enter your marks: 58
You have got B grade

5.4 Switch Case Construct :- When the number of choices after a condition are large, switch case construct is used. Switch Case Construct is a multiway decision construct like if else if construct and performs the same functionality.

Syntax :-

```
Switch (variable)
{
case value 1:
statement block 1
break;
Case value 2:
statement block 2
break;
.....
.....
Case value n:
statement block n
break;
default:
default statement block
break;
}
```



In Switch Case construct, the variable on which the switch has been applied can be a valid integer or character variable

Program to check whether Entered character is Vowel or Consonent :-

```
#include<stdio.h>
#include<conio.h>
Void main()
{
char ch=0;
clrscr();
printf("Enter a character :");
Scanf("%c",&ch);
Switch(ch)
{
case 'a':
printf("Entered character is Vowel \n");
break;
Case 'e':
printf("Entered character is vowel");
break;
Case 'i':
printf("Entered Character is Vowel");
break;
case 'o':
printf("Entered character is Vowel");
break;
Case 'u':
printf("Entered characters is Vowel");
break;
default:
printf("Entered character is Consonent");
}
```

getch();
}

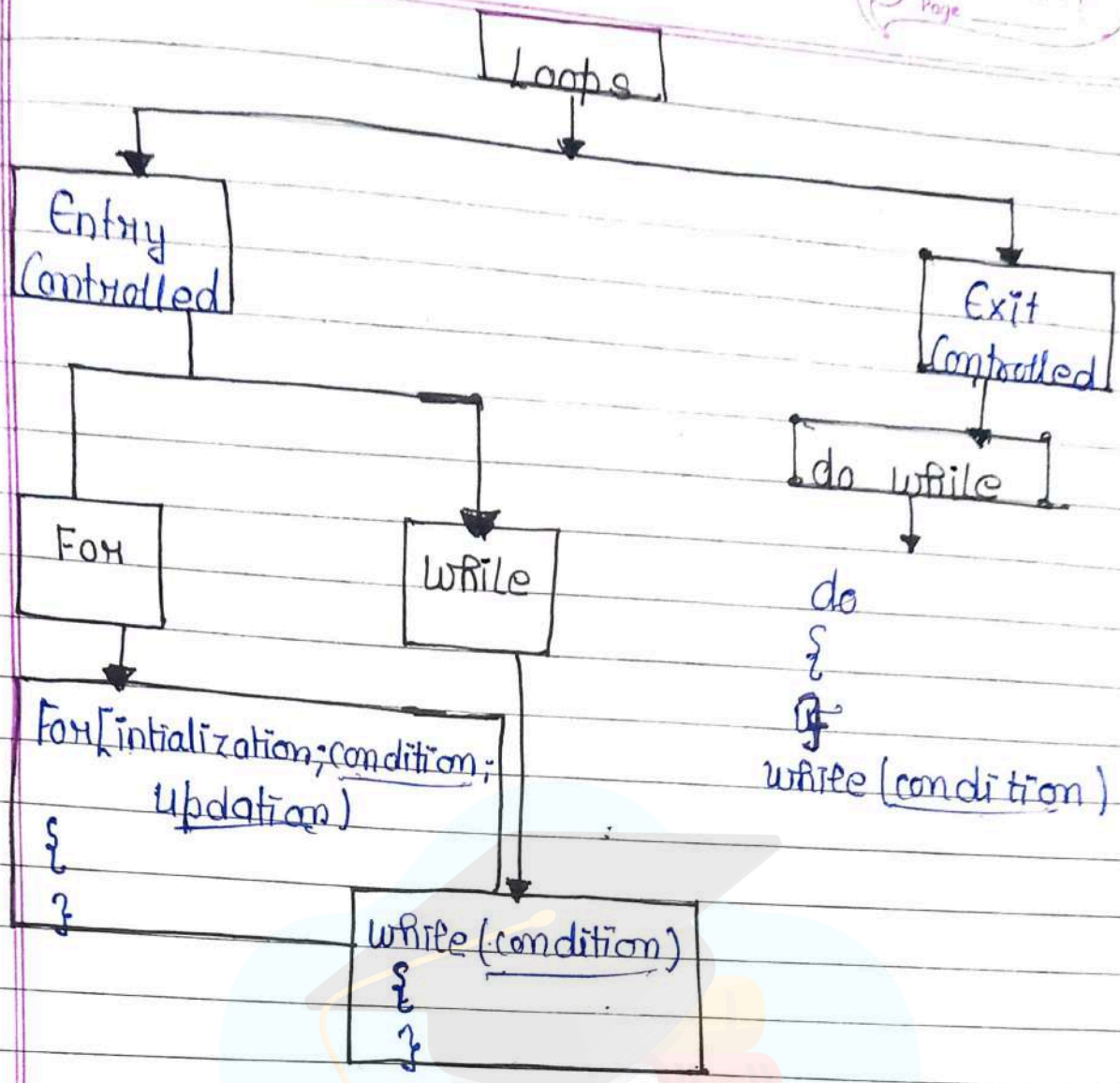
2.4 Iteration or Looping Constructs :- The Concept of repeating the execution of a particular block of statement till a conditional expression is satisfied is called as Iteration or looping. Looping Constructs reduce size of the program files and bothers from programmer's mind.

In loop, the statement needs to be written only once and loop will be executed 10 times as shown below. In Computer programming, a loop is a sequence of instructions that is repeated until a certain Condition is reached

Types of loops :- There are mainly two types of loops :-

1.4 Entry Controlled loops :- In this type of loops the test condition is tested before entering the loop body for loop and while loop are Entry Controlled loops.

2.4 Exit Controlled loops :- In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore the loop body will execute atleast once, irrespective of whether the test condition is true or false do-while loop is exit Controlled loop.



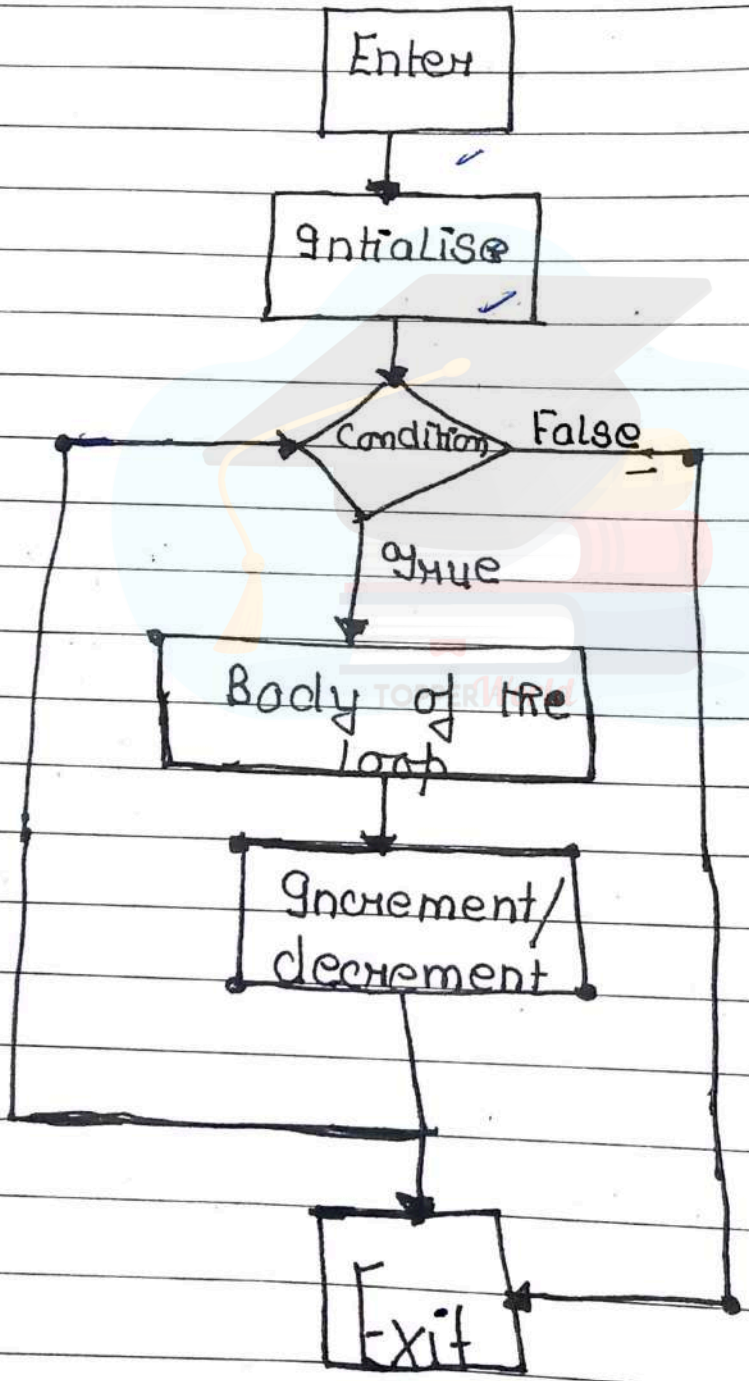
4-] For Loop :- For loop is the most poputar looping Construct. It is very different from while and do while loops. The main advantage of for loop is that it contains all the three phases of loop in single step. The Intialize, condition and Increment phase are contained in single statement. This makes the syntax of this loop a bit simpler than the others. There is no need of Intializa of Conditional vasiable inside or outside the loop body. The Syntax of for loop is :-

Syntax :-

For (intialization phase; condition phase; Increment decrement ph

{
loop body
}

Initialization phase generally consists of an assign-ment, condition phase contains expression that will be tested to carry out looping. The increment or decrement phase increments or decrements the value of the conditional variable. The initia-lization phase is carried out only once in for loop



— Flow chart for Working of for Loop —

Program to Illustrate the working of for loop

```
// Program to print hello world 10 times by using for loop
#include <stdio.h>
#include <conio.h>
Void main()
{
Int i=0;
for(i=1; i<=10; i++)
{
printf("Helloworld \n");
}
getch();
}
```

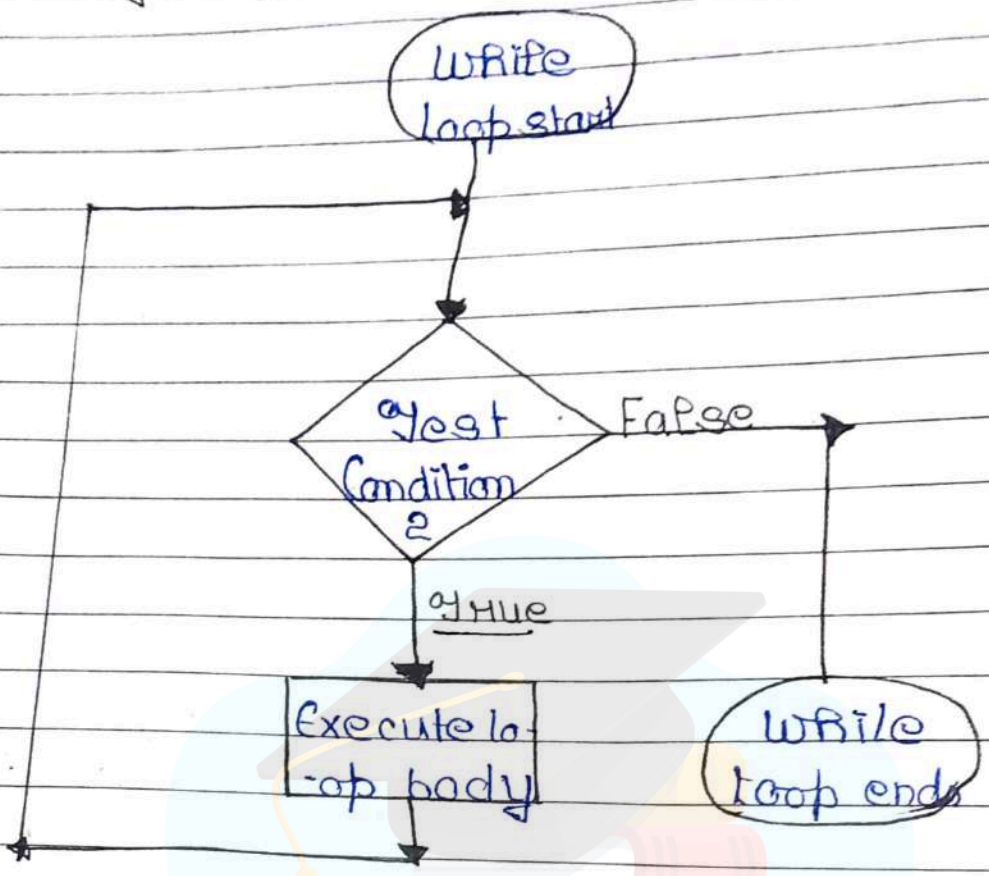
2.4 While Loop :- While studing for loop we have seen that number of [times the] iterations is known beforehand. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we don't know the exact number of iterations of loop beforehand. The loop execution is terminated on basis of test condition.

Syntax :-

```
Intialization expression;
while (test_expression)
{
  // statements
  update_expression;
```

}

Flow diagram/chart :-



Example program :-

```
//Program to print 'hello world' using while loop
#include <stdio.h>
#include<conio.h>
Void main()
{
   int i=1;
   while(i<6).
   {
      printf("Hello world \n");
      i++;
```

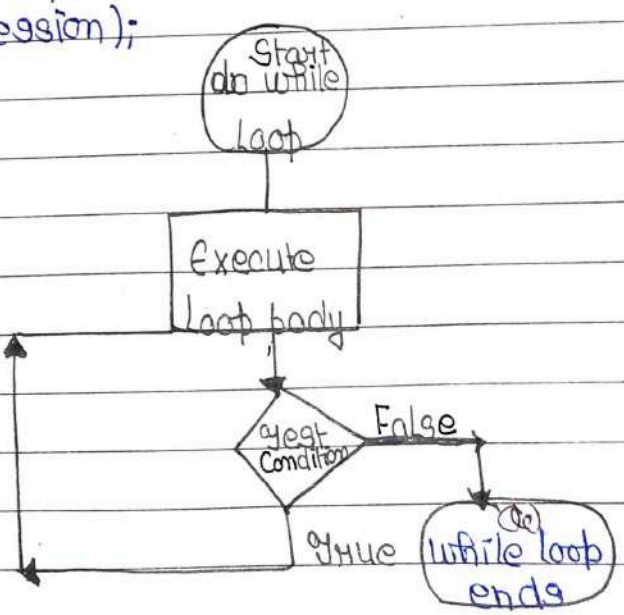
```
    }
    getch();
}
```

3.) Do while loop :- In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body ie do while loop is exit controlled whereas the other two loops are entry controlled loops. In do while loop the loop body will execute at least once irrespective of test condition. do while loop is post tested looping construct in which body of loop is executed first and then condition is tested.

Syntax :-

```
initialization expression;
do
{
  // statements
  update_expression;
} while (test_expression);
```

Flow chart :-



Example :-

```
//c program to illustrate do while loop
#include <stdio.h>
#include <conio.h>
Void main()
{
   int i = 2;
   do
   {
      //Loop body
      Printf("Hello world \n");
      i++;
   }while(i<1);
   getch();
}
```

Illustration :-

In above program test condition (i<1) evaluates to false. But still as the loop is exit Controlled the loop body will execute once

Infinite loop :- An infinite loop is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually this is an error.

3.7 Jumping Statement :- With the use of loop, the execution of certain statements can be repeated for a number of times but we cannot alter the execution sequence of program ie the order in which the different statements of the program are executed. In standard way, the execution of a program starts from the first statement in body of main() function and continues downwards. The conditional constructs can be used to branch the execution on particular path. C language offers a way of altering the sequence of execution. This is with the help of jumping statements

Types of Jumping statements :-

1.7 goto statement
2.7 break statement
3.7 Continue statement

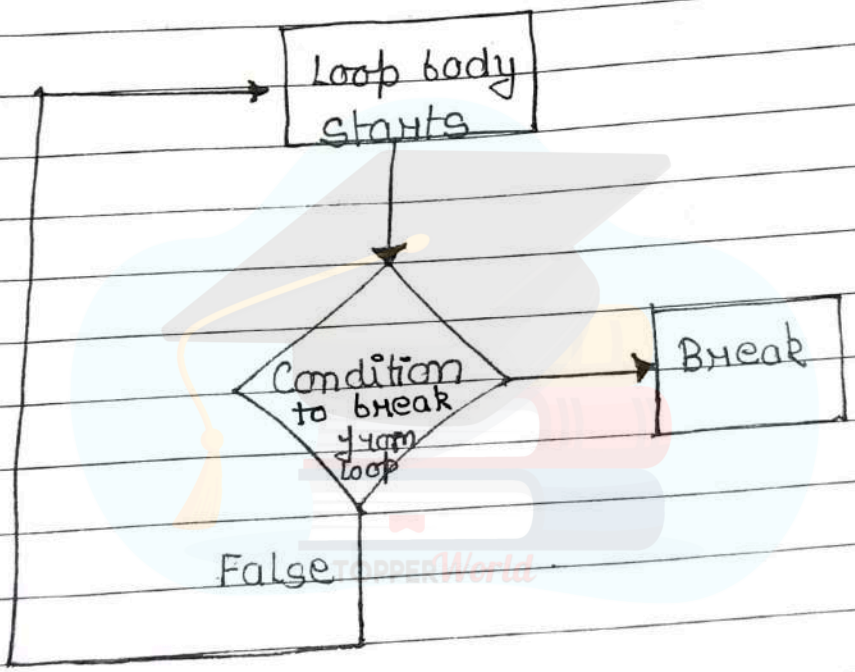
1.7 Break statement :- The Break statement is used to terminate the loops. The termination of the loop can occur in two ways: normal termination and abnormal termination. The normal termination of loop occurs when the condition phase of the loop results in false. In some cases we need to break the execution of ~~loop.~~ [Break] depending on the value input by user. This is called Abnormal termination of loop. break statement is used in

Such Cases. Whenever the break statement is used, the control comes to first statement after loop body. The use of break statement reduces the number of unnecessary executions of loop. This way, it speeds up the program execution. It is used with decision making statement. It forces the loop to stop execution of further iteration.

Syntax :-

```
break;
```

Flow chart :-



Program :-

// Program to print numbers from 1 to 15 and skipping 5 number

```
#include<stdio.h>
#include<conio.h>
Void main()
{
  int i=0;
```

```
clrscr();
For (i = 1; i <= 10; i++)
{
if (i==5)
break;
printf("%d", &i);
}
getch();
}
```

2. Continue Statement:- In some specific programming problems, we need to skip the remaining statements the loop body and take the control to the beginning of the loop. In situation like this. Continue statement is used Continue statement is used to execute other parts of loop while skipping some parts declared inside condition, rather than terminating loop. It Continues to execute the next iteration of same loop. It is used with decision making statement which must be present inside loop. This statement can be used inside for, while or do while loop

Syntax:-

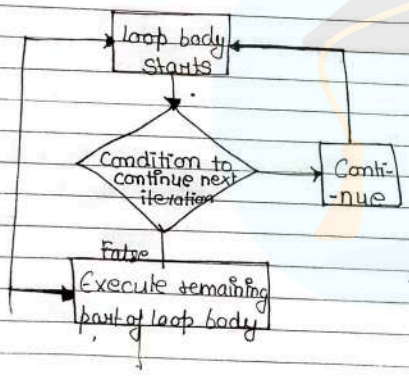
```
Continue;
```

Example:-

```
//Program to print number from 1 to 10 and skipping
5 from 1 to 10 Sequence
#include<stdio.h>
```

```
#include <conio.h>
Void main()
{
int i=0;
clrscr();
for (i=1;i<=10;i++)
{
if (i==5)
Continue;
printf("%d",i);
}
getch();
}
```

Flow chart :-



3.4 Goto statement :- The goto statement is used to alter the sequence of a C program and start execution from a desired Statement. Whenever we use goto statement with a label, the control jumps backward or forward to search the label and executes statements onwards.

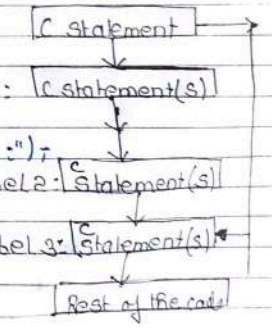
Syntax :-

```
goto label
Statements
Statements
Label: statement
Statement
```

Example :-

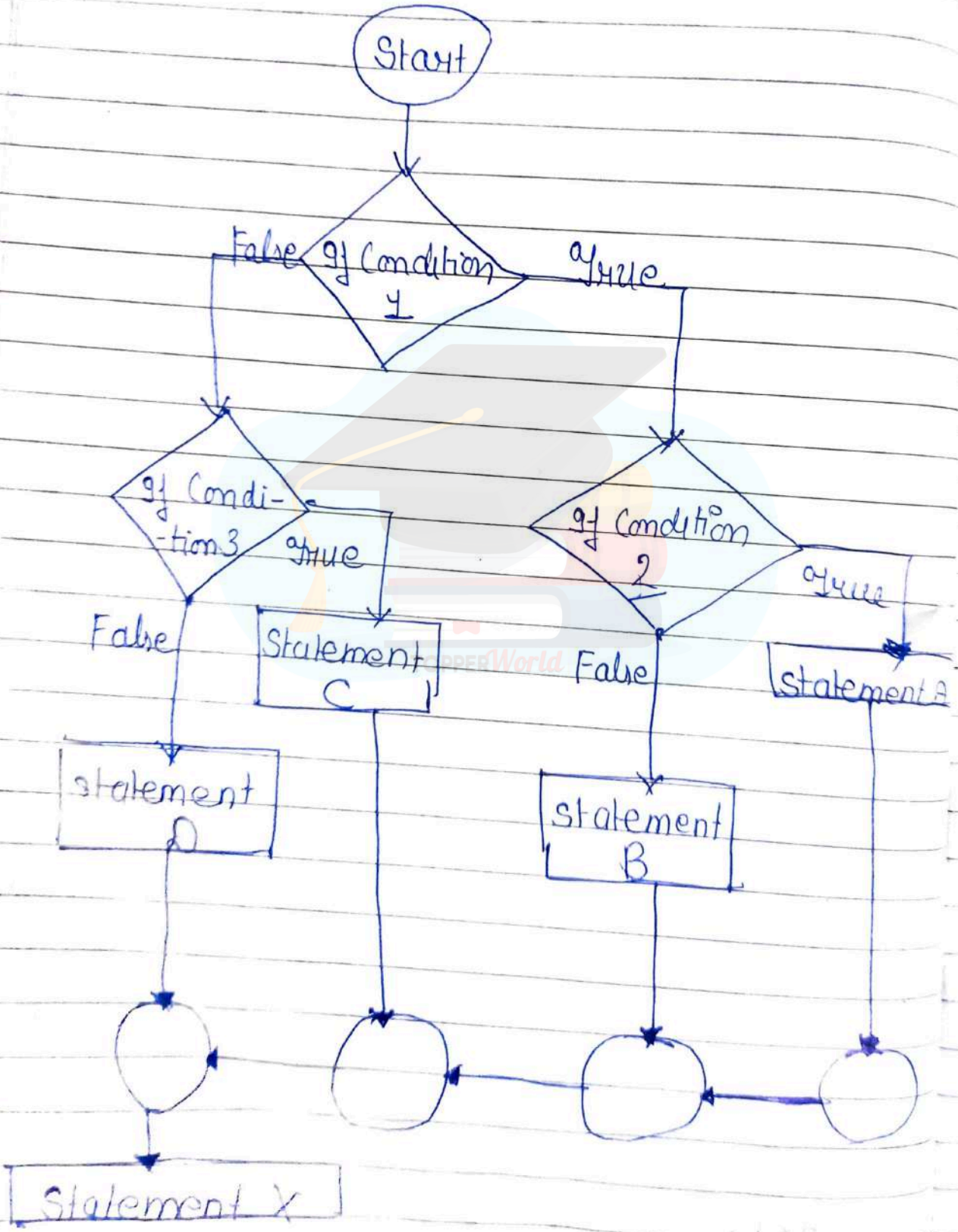
```
//Program to check number is even or odd using goto statement
#include <stdio.h>
#include <conio.h>
Void main()
{
int i;
clrscr();
printf("Enter the value of a :");
scanf("%d",&i);
if(i%2==0)
goto label1;
else
goto label2;
label1:
printf("\n Number is even");
```

Flow chart :- (Goto statement)



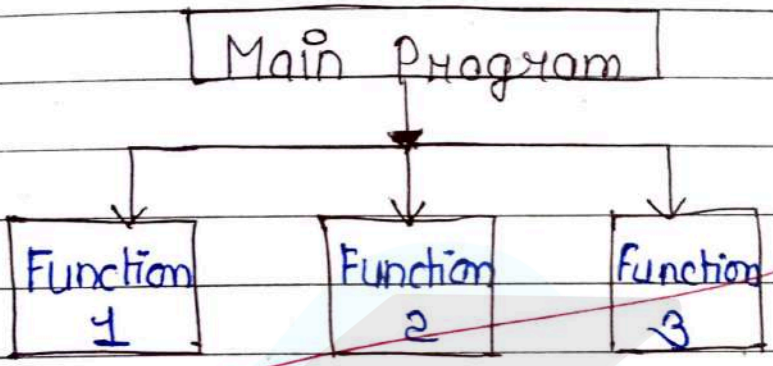
```
Label 2:
puts("\n number is odd");
getch();
}
```

Nested if flow chart :-



Chapter - 6
User defined functions

1. Modularity :- Longer programs are divided into multiple functions, to solve above stated problems. This concept is called as modularity. The main benefit of modularity is parallel development. [The concept can be understood by a practical example]



1. Function :- A function is a set of statements that performs a particular task, whenever called. The function is an independent program that can be called by other functions. Any large C programs can be divided into two or more smaller function. Creating a function is similar to hiring a person and giving him a particular work to perform. The function performs a particular action whenever called. A function is a set of statements written in particular language that performs a particular task, whenever called.

3.7 Types of functions:-

Functions can be broadly categorized into two categories

1.7 Inbuilt or System defined or library functions
2.7 User defined functions.

1.7 Inbuilt functions:- These are the functions whose definition is already defined and stored in respective header files of C library. These functions can be easily incorporated into programs by including the respective header files. Examples of this type of functions are gets() puts(), clrscr(), getch(), printf(), scanf() etc

2.7 User defined functions:- The functions which are created by the user to perform the desired task are called User defined functions.

7 Advantages of User defined functions:-

1.7 No Redundancy/Repitation:- Complex programs generally contain same piece of code at more than one location in a program when we create a function of repeated code, the redundancy of code is reduced, size of the program file is reduced and it becomes easy to design and debug.

2. Universal use :- Some functionality may be needed in more than one programs. There are some common tasks in programs of similar type. When we create a function and make it a part of library, the function is universally available to every program.

3. Modularity :- is a process of dividing a big problems into multiple smaller problems. Modularity is the main benefit provided by functions.

4. Teamwork :- When we divide a problem into a number of functions, the coding can be done in parallel by many people. This reduces the development time and increases the spirit of teamwork in developers

5. Declaration Defining a function :- Declaring a User defined function in c language is very easy. Following is the Syntax of declaring a function in c language.

```
return_type function_name(parameter list){
    body of the function
}
```

Following is the list of elements of a function

1. Return type :- A function may return a value. The return-type is the data

type of the value the function returns. Some function performs the desired operations without returning a value. In this case the return type is the keyword void.

2. Function name :- This is the actual name of the function. The function name and the parameter list together constitute the function signature.

3. Parameters :- A parameter list contains variables that carry information from the main program to function.

4. Function body :- The function body contains a collection of statements that define what the function does.

Example program :-

```
int max(int num1, int num2) {
int result;
if (num1 > num2)
  result = num1;
else
  result = num2;
return result;
}
```

6.] Function Declaration :- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has following parts

return_type function_name(parameter list);

Example for above statement :-

int max(int num1, int num2);

7.] Function Call :- A function call is a special statement, which is written in calling function at a point where the functionality of function is needed function body consists of the actual statements that will be executed by the function. The function from which the call of function is made is called calling function and the function that has been called is called "Called function".

8.] Types of user defined functions :- C allows us to define user defined functions in various way. Th-ese are

1.] function with no argument and no return type
2.] function with no arguments and a return type

3.) Functions with arguments and with no return type
4.) Functions with arguments and with return type

4.) Function with no arguments and no return type :- These are the functions which do not take any data item from calling function in the form of parameters and do not return anything to calling function after execution. As these function do not contain any parameters, a pair of empty braces follow the name of function

Program :-

```
//check from two number that which is bigger
#include <stdio.h>
#include <conio.h>
Void greatnum();
Int main()
{
   greatnum();
   return 0;
}
Void greatnum()
{
  Int i,j;
  printf("Enter 2 numbers that you want to compare...");
  Scanf("%d %d",&i,&j);
  if(i>j){
     printf("The greater number is: %d",i);
  }
```

```
    else {
        printf("The greater number is : %d", j);
    }
}
```

2.] Function with no arguments and a return value :-
This type of function does not take any value from calling function. After execution, they return some value to main function. The data type of returned value determines the return type of function.

Program :-

```
#include<stdio.h>
#include<conio.h>
int greatnum();
int main()
{
  int result;
  result = greatnum();
  printf("The greater number is %d", result);
  return 0;
}
int greatnum()
{
int i, j, greaternum;
printf("Enter 2 numbers that you want to compare.");
scanf("%d %d", &i, &j);
if (i>j) {
  greaternum = i;
```

```
}
else {
greaternum=1;
}
return greaternum;
}
```

3.1 Function with arguments and no return type :-

The function of this type take some value from the calling function and don't return any value from called function from calling function. The values which are passed with the function call are called as parameters or arguments

Program :-

```
#include<stdio.h>
#include<conio.h>
Void greatnum(int a,int b);
int main()
{
 int i,j;
printf("Enter 2 numbers that you want to compare...");
Scanf("%d%d",&i,&j);
greatnum(i,j);
return 0;
}
Void greatnum(int x,int y)
{
 if(x>y){
```

```
        printf("The greater number is %d", x);
    }
    else {
        printf("The greater number is :-%d", y);
    }
}
```

4.4 Function with arguments and return value :-

In this type of user defined functions, the values are passed from calling function to the called function in the form of parameters. The result is calculated in the function. The result is passed back from called function to calling function by using return statement

Program :-

```
#include<stdio.h>
#include<conio.h>
int greatnum(int a, int b);
int main()
{
int i, j, result;
printf("Enter 2 numbers that you want to compare ...");
scanf("%d %d", &i, &j);
result = greatnum(i, j);
printf("The greatnumbes is %d", result);
return 0;
}
int greatnum(int x, int y)
```

```
{
if (x>y) {
  return x;
}
else {
    return y;
}
}
```

9.) Actual parameters :- The variables used to pass information from calling function to the called function are called Actual parameters

10.) Formal parameters :- The variables created in the function body to hold the values of actual parameters are called formal parameters

11.) Passing Parameters to function :- As we know that parameters are passed from calling function. We can call a [method] function in two different ways

1.) Call by value method
2.) Call by reference method

1.) Call by value method :- In Call by value method a copy of actual argument created and passed to formal arguments in the function definition. Thus for each and argument, A copy of value is created and

passed to formal argument. That is why called "Call by Value method".

2. Call by reference method :- In this method of calling a function, the address of actual arguments is passed, instead of creating the copy of actual argument. The called function makes changes directly to actual parameters and no copies of parameters are created. This process increases the efficiency of programs.

Program :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int sum(int *, int *);
 int a,b, ans;
 printf("\n Enter 2 numbers");
 scanf("%d %d",&a,&b);
 ans = sum(&a,&b);
 printf("\n The result after addition is %d",ans);
 getch();
}
 int sum(int *j1, int *j2)
 {
 int result;
 result = *j1 + *j2;
 return(result);
}
```

12. Recursion :- Recursion is a technique that involves a function calling itself from its body. Recursive function is a function that calls itself from its own body. The function keeps on calling itself till a particular condition holds true

Program to Calculate factorial of a Number Using recursion :-

```
#include<stdio.h>
#include<conio.h>
int fact(int);
int main()
{
 int n,f;
 printf("Enter number whose factorial you want to calculate");
 scanf("%d",&n);
 F=fact(n);
 printf("factorial= %d",f);
}
int fact(int n)
{
 if(n==0)
 {
  return 0;
 }
 else if(n==1)
 {
  return 1;
 }
 else
```

```
    {
        return n*fact(n-1);
    }
}
```

Advantages :-

1. Recursion makes the program code compact
2. For some complicated problems, recursion can lead to solutions that are much cleaner and easier to write and modify.
3. Recursion is very simple and suitable to be used for data structures like stacks, queues, trees etc.
4. In case of recursion system takes care of internal stack.
5. Recursion is useful for the problem in which there is some repetition.

Limitations :-

1. Recursion is slower in terms of speed and execution time.
2. Recursion takes more memory space as variables

13.) Local Variables :- Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. [All the variables a, b, c are local to main() function

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main(){
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf("Value of a = %d, b = %d and c = %d \n", a, b, c);
}
```

14.) Global Variables :- Global variables are defined outside a function, usually on top of the program. Global Variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program. A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Example :-

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

```
int g;
Void main()
{
 int a,b;
 a = 10;
 b = 20;
 g = a+b;
 printf("Value of a = %d, b = %d and g = %d \n", a,b,g);
}
```

Storage Classes in C :- Storage classes in C are used to determine the life-time, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

1. Automatic
2. External
3. Static
4. Register

	Storage classes	Storage Place	Default Value	Scope	Lifetime
1.	Auto	RAM	Garbage Value	local	within function
2.	Extern	RAM	Zero	Global	Till end of main program maybe declared anywhere in program
3.	Static	RAM	Zero	Local	Till end of main program, Retains value between multiple functions Call

4.	Register	Register	Garbage Value	Local	within the function

3. Automatic :- Automatic variables are Allocated memory automatically at runtime. The Visibility of automatic Variables is limited to the block in which they are defined. The scope of the automatic Variables is limited to the block in which they are defined. The automatic Variables are intialized to garbage by default. The memory assigned to automatic Variables gets freed upon exiting from the block. The keyword used for defining automatic variables is auto. Every local variable is automatic in C by default

Example :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
  int a;
  char b;
  float c;
  printf(" %od %oc %of ",a,b,c);
}
```

4. Static :- The Variables defined as static specifier can hold their value between the multiple function calls. Static local variables are visible only to the function or the block in which they are defined. A same static Variable can be declared

many times but can be assigned at only one time.
Default intial value of static integral variable is
0 otherwise null. The visibility of the static glo-
-bal variable is limited to file in which it has
declared. The keyword used to define static variab
is static.

Syntax :-
Example :- Static data-type variable_name;

```
#include<stdio.h>
#include<conio.h>
Static char c;
Static int i;
Static float f;
Static char s[100];
Void main()
{
  printf("%d%d%f%s",c,i,f);
}
```

3.) Register :- The variable defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in CPU. We can not dereference the register variables ie we cannot use & operator for register variables. The access time of register variables is faster than automatic variables. The intial default value of register local variable is 0. The "register keyword is used for variable which should be stored in CPU register. We can store pointers into register. [A register can store address of Variables. Static variable can not be

stored into the register also. We can not use more than one storage specifier for same variables

Syntax :-
register data_type variable_name

Example :-
```
#include<stdio.h>
#include<conio.h>
int main()
{
 register int a;
 printf("%d",a);
}
```

4.4 External :- The external storage class is used to tell the Compiler that the variable defined as extern is declared with an external linkage elsewhere in program. The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in program the default initial value of external integral type is 0 otherwise Null. We can only initialize the extern variable globally. We can not initialize external variable within any block or method. An external variable can be declared many times but can be initialized at only one. If a variable is declared as external then Compiler searches for that variable to be initialized somewhere in program which may be extern or static. If it is not, then Compiler will show an error.

Example :-
```
#include<stdio.h>
```

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

It will print error

Example:-

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

Syntax:-

extern data_type variable_name

Example:-

```
#include<stdio.h>
#include<conio.h>
Extern int gl=10;
Void main()
{
  void f1();
  printf("\n the value of gl is = %d",gl);
  f1();
  printf("\n the value of gl after calling f1() is %d",gl);
  getch();
}
  void f1()
  {
    gl=gl+5;
  }
```

Chapter - 7
Array

1) Array :- An array in C is a Collection of similiar data items stored at Contiguous memory locations and elements can be Accessed randomly using indices of an array. They Can be Used to store collection of primitive data types such as int, float, double, char etc

Declaration of Array :- There are different ways to declare Arrays in C language
To declare an array, we must provide following information
1.) Data type of an array
2.) The name of the array
3.) The number of subscripts in an array
4.) The maximum value of each subscript
like other Variables in C, an array has to be declared before it is used. Following is the Syntax for declaring an array
<Datatype> <array name>[n]

Example :-
int student[50]
Declares an array student of integer data type that can store 50 elements. That means, it is equivalent to 50 different variables of integer type. Each element in the array has a index number which starts from 0 and goes upto n-1 Each member can be addressed as student[0], student[1], student[2] and so on. The

Index number of array always starts from 0 and not from 1.

Intializing Arrays :- We can intialize an array in C either one by one or using a single statements as follows :-

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces cannot be larger than the number of elements that we declare for the array between square brackets [ ]

If you omit the size of array an array just big enough to hold intialization is created.

double balance[ ] = {1000.0, 2.0, 3.4, 7.0, 50.0};

We can assign a single element of array

balance[4] = 50.0;

Accessing Array :- A single dimensional or linear array is accessed by using a loop by which a variables is used as the index number of array. The index variable intialized to 0 at beginning of the loop and it is incremented to a number, one lesser than the size of array. for example array ar[5] has the elements ar[1], ar[2], ar[3], ar[4]. We can access these elements through a variable 'i' whose value starts from 0 and goes upto 4. So a single statement ar[i] replaces the individual elemen

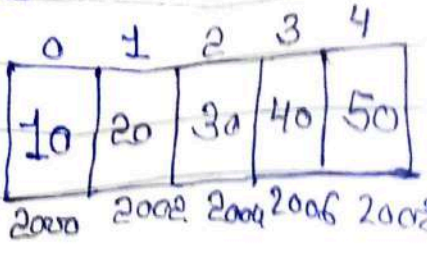
names of array. In each iteration of Loop the value of "i" is incremented and next element of array is accessed.

Program :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
 int ar[5];
 int i;
  for (i=0; i<5, i++)
       {
        printf("\n Enter Array element : ");
        scanf("%d", &ar[i]);
       }
  printf("\n The entered Values are : ");
       for (i=0; i<5; i++)
       {
        printf("%d", ar[i]);
       }
getch();
}.
```

One dimensional array :- Once array is defined the each element of an array can be accessed by Array name followed by an index enclosed in brackets. All array elements are numbered starting from 0

Eg : int S[5] = {10, 20, 30, 40, 50};



2.7 Two dimensional array in C :- The two dimensional array can be defined as an array of arrays. The 2 dimensional array is organized as matrices which can be represented as collection of rows and columns. However 2d arrays are created to implement a relational data base look like data structure. In 2d array each element can be accessed simply by array name followed by two index numbers enclosed in brackets.
Eg : int a[2][3] = {10,20,30,40,50,60}

Declaration of two dimensional Array in C :-
The syntax to declare 2D array is given below

data_type array_name[rows][columns];

Example :-

int twodimen[4][3];

Here 4 is the number of rows and 3 is the number of columns

Initialization of 2D array in C :- In single dimensional array, we don't need to specify the size of array if declaration and initialization are being done simultaneously. However this will not work with 2d arrays. We will have to define atleast the second dimension of array. The 2d array can be declared and defined as :

int arr[4][3] = {{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Example of Two dimensional area :-

```
#include<stdio.h>
#include<conio.h>
Void main()
{
 int a[5][4];
 int i;j;          //input
 for(i=0;i<5;i++)
 {
    for(j=0;j<4;j++)
    {
      printf("\t Enter the element");
      Scanf("%d",&a[i][j];
    }
 printf("\n");
}
printf("\n You entered \n");  //output
 for(i=0;i<5;i++)
 {
    for(j=0;j<4;j++)
    {
      printf("%d \t";j<4;j++)
    }
 printf("\n");
}
getch();
}
```

24 Program to add two matrix into third matrix:

```
#include <stdio.h>
#include <conio.h>
Void main()
{
  int ary1[4][4] = {
                    {1,2,3,4}
                    {1,2,3,4}
                    {1,2,3,4}
                    {4,2,3,4}
                  };
int ary2[4][4] = {
                    {1,2,3,4}
                    {1,2,3,4}
                    {1,2,3,4}
                    {1,2,3,4}
                  };
int ary3[4][4];
int i,j;
clrscr();
//input
for(i=0;i<4,i++)
{
   for(j=0;j<4;j++)
   {
     ary3[i][j] = ary1[i][j] + ary2[i][j];
   }
}
printf("\n the array after addition is \n");
```

matrix:

```
//output
for(i=0; i<4; i++)
{
  for(j=0; j<4; j++)
  {
    printf("%d \t", arr3[i][j];
  }
  printf("\n");
}
getch();
}
```

Passing an array as parameter to function :- We can pass an entire array as an argument or parameter to a function this can be done by using two methods

1. Passing array as an argument using Subscript notation
2. Passing array using pointer notation

1. Subscript notation method :- In the subscript notation method we pass the array name along with a variable to hold the size of the array. Both of them are used as actual parameters. The similar declaration done for formal parameters. In the function prototype and function definition, a pair of empty square braces follow the name of the array but in the function call, only the names of the array and size is passed.

Example :-

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

```
Void main()
{
 int arr[5], i;
 int max(int temparr[], int size);
   printf("\n Input element in array");
     for(i=0; i<5; i++)
     {
       printf("\n Enter the value");
       scanf("%d", &arr[i]);
     }
   printf("\n The largest element array is %d", max(arr,5));
   getch();
}
 int max(int temparr[], int size)
 {
   int m=0; int i;
   for(i=0; i<size; i++)
   {
     if(temparr[i]>m)
     {
     m = temparr[i];
   }
 }
 return(m);
}
```

Strings in C :- The string can be defined as the one dimensional array of characters terminated by null ['\0']. The character array or string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory and last character

must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with null in memory

Declaration :- There are two ways to declare a string in C language.

1. By char array
2. By string literal

Example :-

char ch[11] = {'j', 'a', 'v', 'a', 'S', 'h', 'a', 'R', 'M', 'A', '\0'}

We know that array index starts from 0, so it will be represented as :-

0	1	2	3	4	5	6	7	8	9	10
j	a	v	a	S	h	a	R	M	A	\0

While declaring string, size is not mandatory. So we can write the above code as

String literal :-

char ch[] = {"Javasharma"};

char ch[] = {'j', 'a', 'v', 'a', 'S', 'h', 'a', 'R', 'M', 'A', '\0'};

String Example in C :-

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
char ch[11] = {'j', 'a', 'v', 'a', 'S', 'h', 'a', 'R', 'm', 'a', '\0'};
```

```
char ch2[11] = "javatpoint";
printf("Char Array Value is %s \n", ch);
printf("string Literal value is %s \n", ch2);
return 0;
}
```

Traversing String :- Traversing the string is one of the most important aspect in any of programming language. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is sometimes different from traversing an integer array. We need to know the length of array to traverse an integer array whereas we may use the null character in case of string to identify the end the string and terminate the loop.

There are two ways to traverse a string

1. By using the length of string
2. By using the null character.

C gets() function :- The gets() function enables the User to enter some characters followed by the enter key. All the characters entered by user get stored in characters array. The gets() allows the user to enter the space separated strings.

Declaration :- char[] gets(char[]);

Reading String using gets()

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

```
Void main()
{
  char str[30];
  printf("Enter the string ");
  gets(s);
  printf("You entered %s",s);
}
```

C Puts function :- Puts() function is very much similar to printf function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. Puts function prints an additional newline character with string which moves the cursor to the new line on console.

Declaration :-

```
int puts(char[]);
```

Example :-

```
#include<stdio.h>
#include<string.h>
int main(){
char name[50];
printf("Enter your name:");
gets(name);
printf("Your name is :");
puts(name);
return 0;
}
```

String Manipulation functions :- In addition to input and output functions, C also has many string manipulation functions

No	Function	Description
1.	strlen(string-name)	Returns the length of string name.
2.	Strcpy(destination, source)	Copies the contents of source string to destination string
3.	Strcat(first_string, second_string)	Concats or join first string with second string. The result of the string is stored in first string.
4.	Strcmp(first string, second-string)	Compares the first string with second string. If both strings are same it returns 0
5.	Strrev(string)	returns reverse string
6.	Strlwr(string)	returns string characters in lowercase
7.	Strupr(string)	returns string characters in uppercase

1. Strlen() :- [C string length]

The strlen() function returns the length of the given string. It does not count null character '\0'.

Syntax :- strlen(array_name);
[variable_name];

Example :-

```
#include<stdio.h>
#include<string.h>
Void main(){
char ch[20] = {'s','h','u','b','h','a','m','\0'};
    printf("Length of string is %d", strlen(ch));
}
```

2.] Strcpy() :- [C copy string]

The Strcpy(destination source) function copies the source string in destination

Example :-

```
#include<stdio.h>
#include<string.h>
Void main(){
char ch[20] = {'s','h','U','B','H','A','M','\0'};
char ch2[20];
Strcpy(ch2,ch);
printf("Value of Second string is %s",ch2);
}
```

Syntax :-

Strcpy(destination, source);

3.] Strcat() :- [C string Concalenation]

The Strcat(first_string, second_string) function Concalenate two strings and result is returned to first_string.

Example :-

```
#include<stdio.h>
#include<string.h>
Void main(){
char ch[10] = {'h','e','L','L','o','\0'};
char ch2[10] = {'c','\0'};
Strcat(ch,ch2);
printf("Value of first string is %s",ch);
}
```

Syntax :-

Strcat(first string_second string)

4.] Strcmp() :- [C compare string]

The Strcmp(first_string, second_string) function compares two string and returns 0 if both Strings are equal. In below example we are using gets() function which reads string from Console

Example :-

```
#include <stdio.h>
#include <string.h>
Void main(){
char str1[20],str[20];
printf("Enter 1st string:");
gets(str1);
printf("Enter 2nd string:");
gets(str2);
if(strcmp(str1,str2)==0)
    printf("strings are equal");
else
    printf("strings are not equal");
}
```

5. strrev() :- [c Reverse string]

The strrev(string) function returns reverse of the given string :-

Example :-

```
#include <stdio.h>
#include<string.h>
Void main(){
char str[20];
printf("Enter string:");
gets(str);    // reads string from Console
printf("string is : %s",str);
printf("\n Reverse string is %s",strrev(str));
}
```

6. strlwr() :- [c String lowercase]

The strlwr(string) function returns string characters in lowercase- Syntax :- strlwr(string name);

Example 6-

```
#include <stdio.h>
#include <string.h>
Void main() {
char str[20];
 printf("Enter string:");
 gets(str); //reads string from Console
 printf("string is %s",str);
 printf("\n lower string is %s", strlwr(str));
}
```

7.7 strupr() 6- [C string Uppercase]

The strupr(string) function returns string characters in uppercase.

Example 6-

```
#include<stdio.h>
#include<string.h>
Void main() {
char str[20];
printf("Enter string:");
gets(str);
printf("string is %s",str);
printf("\n Upper string is %s", strupr(str));
}
```

Chapter - 8
Pointers

1. C Pointers :- A pointer is a variable that stores the address of some other variable. It is a derived data type that is created from the fundamental data types. The pointer variable can also access the value stored in the variable whose address they contain. A pointer is a variable that is stored at a location in a memory and it stores the address of variable to which it points if

Example :- If a variable "Var" having value 100 has been stored at memory location address 6000 in memory the pointer to "Var" will contain a value 6000. A pointer to "Var" can manipulate the value stored at location of var ie 100.

Variable	Value	Address
Var	100	6000
P	6000	4048

2. Advantages of Pointer :- The advantages of pointer are:

1. With the use of pointers, one can return multiple values from function.
2. We can allocate or deallocate space in memory by using pointers
3. Pointers are used to efficiently handle the arrays
4. Pointers are used to implement several data stru

like stacks, Queues, Linked, lists and trees.

3] By using pointers, one can pass an array or a string as a parameter to a function

4] The use of pointers results into faster and more effective code

5] Address of (&) operator :- The address of" operator returns the address of the variable to which it precedes. We can mostly use this operator while using Scanf() function

Example :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
  int var = 100;
  printf("\n The address of Var is %u", &Var);
  getch();
}
```

4] Null pointer :- A pointer that is not assigned any value but Null is known as the null pointer. If we donot have any address to be specified in the pointer at the time of declaration we can assign Null value

Example :-

```
int *p = NULL;
```

Exc

5:] Pointer Program to swap two numbers without Using third Variable

```
#include <stdio.h>
Void main(){
int a=10,b=20,*p1=&a,*p2=&b;
printf("Before Swap:*p1=%d *p2=%d",*p1,*p2);
*p1 = *p1+*p2;
*p2 = *p1-*p2;
*p1 = *p1-*p2;
printf("\n After Swap:*p1=%d *p2=%d",*p1,*p2);
}
```

6:] Pointer Variables Declaration :-

A pointer variable has to be declared before it can be used in program.

Syntax :-

```
Data_type *name;
```

Here "data type" is the type of the pointer variable and "name" is the name of pointer Variable. The *[Asterisk] is used to inform the Compiler that the variable being declared is a pointer.

If we don't intialize pointers, they will have garbage Values.

7:] Intializing pointers :- Pointers are also intialized by using assignment operator (=) of C. Pointers are intialized with addresses.

Example :-

```
int a = 90;
int *ptr = &a;
```

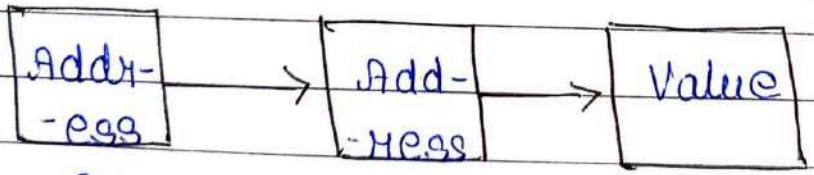
A pointer can point to the Variable having same type

as the pointer Variable.

Using pointer in C :-

Pointers can be used to access the address as well as the data of variable whose address it contains. To access address, the name of pointer variable is used, whereas to access the data stored in variable pointed by pointer, * is preceded with name of pointer variable. *[Asterisk] instructs the compiler that variable being declared is a pointer variable

Pointer to Pointer [C Double Pointer] :- In C language we can also define a pointer to store the address of another pointer Such pointer is known as double pointer. The first pointer is used to store address of variable whereas second pointer is used to store address of first pointer



Syntax :-

```
int **p;
```

Example :-

```
#include<stdio.h>
Void main() {
int a = 10;
int *p;
```

```
int **pp;
p = &a;
pp = &p;
printf("Address of a: %x", p);
printf("Address of p: %x", pp);
printf("Value stored at p: %d", *p);
printf("Value stored at pp: %d", **pp);
}
```

Pointer Arithmetic in C & C++ Language allows you to perform arithmetic operations on pointer. The result of an arithmetic operation performed on pointer will also be a pointer if other operator is of type integer. In pointer from pointer subtraction the result will be an integer value. We can perform following arithmetic operators on pointer in C:

1. Increment
2. Decrement
3. Addition
4. Subtraction
5. Comparison

1. Incrementing Pointer in C & C++ If we increment a pointer by 1, the pointer will start pointing to immediate next location. This is somewhat different from general arithmetic since value of pointer will get increased by size of the data type to which pointer is pointing.

The Rule to Increment pointer is :

new_address = current_address + i * size_of(data type)

Example :-

```
#include<stdio.h>
Void main(){
int number=50;
int *p;
p=&number;
printf("Address of p variable is %u";p);
p=p+1;
printf("After increment address of p variable is %u",p);
}
```

Traversing an array by using pointer :-

```
#include<stdio.h>
Void main()
{
  int arr[5]={1,2,3,4,5};
  int *p=arr;
  int i;
  printf("Printing array elements...");
  For(i=0;i<5;i++)
  {
    printf("%d",*(p+i));
  }
}
```

2. Decrementing Pointer in C :- We can decrement a pointer variable. If we decrement a pointer, it will start pointing to previous location.

Rule :-

new_address= current_address - i * size_of(data type)

Example :-

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

```
Void main(){
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u ",p);
p = p-1
printf("After decrement address of p variable is %u",p);
}
```

3.4 Addition pointer in C :- We can add a value to the pointer variable the formula to add addition pointer is below :-

new_address = current_address + (number * size_of(data type))

Example :-

```
#include<stdio.h>
Void main(){
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u",p);
p = p+3;
printf("After adding 3 Address of p variable is %u \n",p);
}
```

4.4 Subtraction pointer in C :- In C we can subtract a value from pointer variable subtracting any number from pointer will give an address. The formula of subtracting value from pointer variable is

new_address = current_address - (number * size of(datatype))

Example :-

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

```
Void main() {
int number = 50;
int *p;
p = &number;
printf("Address of p variable is %u", p);
p = p-3
printf("After Subtracting 3 Address of p variable is %u", p);
}
```

Illegal arithmetic with pointers :-

Address + Address = illegal
address % address = illegal
address * address = illegal
address / address = illegal
address & address = illegal
address ^ address = illegal
address | address = illegal
~ address = illegal

Pointer to function in C :- A pointer can point to a function in C. However the declaration of pointer variable must be the same as function

Example :-

```
#include <stdio.h>
int addition();
int main()
{
  int result;
  int (*ptr)();
  ptr = &addition;
```

```
    result = (*ptr)();
    printf("The Sum is %d", result);
getch();
}
int addition()
{
    int a, b;
    printf("Enter two numbers");
    scanf("%d%d", &a, &b);
    return a+b;
}
```

Array of Pointer in C :- Pointers are also used to handle array of string or two dimensional character array. In it each row is handled be a pointer with unique index number. A single for loop is used to access the contents of array of pointer.

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main() {
char* str[3] = {"Microsoft", "Oracle", "Sun"};
int i;
    for (i=0; i<3; i++)
    {
        printf("The element is :");
        puts(str[i]);
    }
getch();
}
```

Pointers as Arguments :- In call by reference address of actual parameters is passed and parameters are directly manipulated without creating an copy. This is done with the help of pointers.

Program of use of pointers while calling a function :-

```
// Program to illustrate call by reference
#include <stdio.h>
#include <conio.h>
Void main() {
 int sum(int*, int*);
 int a, b, ans;
 printf("Enter two numbers");
 scanf("%d%d", &a, &b);
 ans = sum(&a, &b);
printf("\n The result after addition is %d", ans);
getch();
}
 int sum(int *j1, int *j2)
 {
   int result;
   result = *j1 + *j2;
   return (result);
 }
```

Function returing pointers :- We can also create function that return pointer variables with this method we can return multiple values from function depending on some condition.

Program Using functions returning pointers :-

```
#include<stdio.h>
#include<conio.h>
int *great(int*, int*);
void main()
{
  int a,b;
  int *ans;
  printf("\n Enter 2 numbers");
  scanf("%d%d",&a,&b);
  ans = great(&a,&b);
  printf("The biggest number is %d", *ans);
  getch();
}
int* great(int *F1, int *F2)
{
  if(*F1>*F2)
   return(F1);
   else
    return(F2);
}
```

2.) Dynamic Memory Allocation :- Memory allocation means to reserve the block of memory for some variable. If we declare array in static declaration

Eg int a[10][10];

In this case 10x10 = 100 elements of 2 bytes each thus 200 bytes of memory is allocated to the variable a. But at run time we require 3x3 matrix which means Nine[9] elements of 2 bytes each

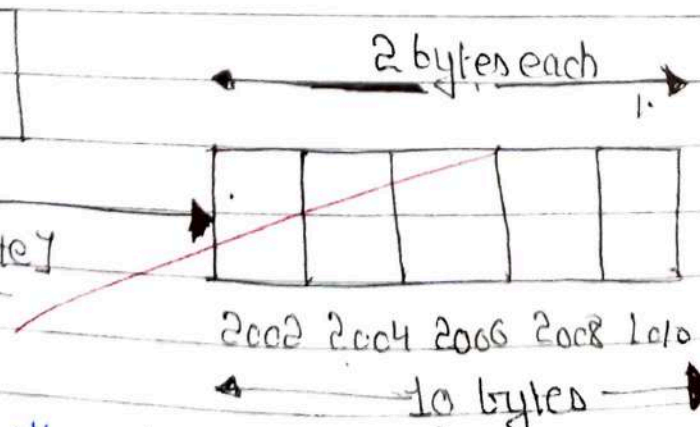
only 18 bytes are allocated So 200 - 18 = 182 bytes
It means 182 bytes are waster. So solution of this problem is to allocate memory at runtime [allocation]
In such type of memory allocation is called as dynamic memory allocation. In dynamic memory allocation we can allocate memory to variables during execution time. For dynamic allocation we use four functions malloc(), calloc(), free() and realoc()

1. malloc() :- The malloc() function reserves a block of memory of specified size and returns a pointer of specified datatype. Malloc() is dynamic memory allocation function.

Eg p=(int *)malloc(5*size of(int));

Here p is the pointer of type int.
(Int*) tells to what type it will be pointing.
malloc is memory allocation function
5 is the number of elements to be allocated

Size of(int) is used to tell the size of particular datatype



In fig malloc function reserves a memory equal to 10 bytes and address is stored in p.

2. Calloc() :- It also work like malloc() function. It allocates space for elements, initialize them to zero and then returns a pointer to the memory calloc is also a dynamic memory allocation function.

eg: p = (int *) calloc(5 * sizeof(int));

In this example calloc function reserves a memory equal to a 10 bytes. It also initializes all the 5 elements to zero and returns a pointer in p.

3. Free() function :- This function deallocates or frees a block of memory that was previously allocated. This function takes a pointer to block of memory as its argument and after its execution that block is no more availble.

Syntax :-

```
int *p;
free(p);
```

4. Realloc() function :- This function is used to modify the size of previously allocated space

eg:- if p = (int *) malloc(5);