2. ALGORITHMS

2.1 Algorithms –

- An algorithm is a list of instruction for accomplishing a task. When algorithm is executed it transforms information from input into output. The instructions are a finite set of steps that can be executed in a specific order. When these steps are executed, the execution must terminate after a finite time.
- Algorithm is expected to terminate and is expected to provide answer. (The answer may also be – there is no answer). For example, if we write an algorithm to calculate factorial of a number and if the accepted number is negative then the algorithm's answer will be there is no answer.
- Different ways to express an algorithm are,
  a) Step form
  b) Pseudo code
  c) Flowcharts
- In **step-form** representation, the procedure of solving problem is started with written statement. Each statement solves a part of the problem and all steps together solve a complete problem. The step-form algorithm uses a normal language. The steps will have logical relation with preceding steps.
- The **pseudo-code is** almost like a **step-form** representation. It is written form of the algorithm with more precision using limited vocabulary
- The flowchart representation of the algorithm is the pictorial representation of an algorithm using typical symbols for different processes. The flowchart shows the overall flow of the solution. This format is better when the algorithms are bigger in size. The pictorial representation makes it easy to understand because it almost becomes self explanatory.

2.2 One Practical Example of an Algorithm –

Algorithm to go from Ulhasnagar to a shopping mall located in Thane –

1. START.
2. IF not **well dressed** up THEN Get ready properly.
3. Start from home.
4. IF you need Auto Rickshaw THEN Hire it upto the Railway station
   ELSE Go waking upto Railway station.
5. IF you do not have a proper pass or ticket THEN Buy a pass / ticket
6. Come to appropriate platform.
7. IF train is not yet arrived THEN GOTO step 7.
8. Get into the train.
9. Get down at Thane station.
10. IF friend(s) is / are not at the decided place Then GOTO step 10.
11. IF you need Auto Rickshaw THEN Hire it upto the Shopping Mall.
    ELSE Go walking upto Shopping Mall.
12. Enter into the Shopping Mall.
13. STOP.

Explanation –

➢ The above example indicates that algorithm shows three features.

1) Sequence (Process) – Steps are executed in the simple manner. (Steps 1, 3, 6, 8, 9, 12, 13)
2) Decision (Selection) – The decision making can be done in two ways.
   (a) IF THEN – Step 5, 9
   (b) IF THEN ELSE – Steps 4, 11.
3) Repetition (Iteration or Looping) – The repetition can be implemented with three different ways
   a) Using GOTO – Step 7, 10.
   b) Using –

```
Repeat
    .....
    .....
    .....
Until (condition)
```

   c) Using –

```
While (condition)
Begin
    .....
    .....
    .....
End
```

2.3 Algorithms Development –

Following are some important points in developing step form algorithms.

1. Each algorithm will be enclosed between the two words **START** and **STOP.**
2. To accept data from user, **INPUT** or **READ** statements are used.
3. To display any message to the user or to display the value of a variable, **PRINT** statement will be used.
4. There can be several simple steps in the algorithm which are to be executed. These steps must be mentioned in the proper order.
5. There can be number of different operations which are to be executed. These operations will be indicated using appropriate operators.

a) Assignment Operator

y ← 10

which means that value 10 is assigned to the variable y.

b) Arithmetic Operators

The typical operators used are,

+	addition	-	subtraction
*	multiplication	/	division

d) Relational Operators

>	greater than	>=	greater than or equal to
<	less than	<=	less then or equal to
=	equal to	!=	not equal to

e) Logical Operators

AND→Logical AND
OR →Logical OR
NOT→Logical NOT

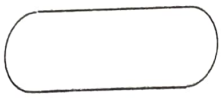
2.4 FLOWCHARTS –

The Flowchart gives pictorial representation of the algorithm.

The pictorial representation makes it self explanatory.

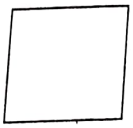
The Flowcharts are more useful when the algorithms are large in size.

Following are the symbols which are typically used to develop the algorithms

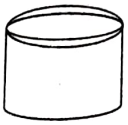


Start or end of the program or flowchart

A magnetic tape

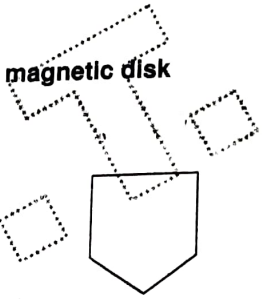


Computational steps or processing function of a program

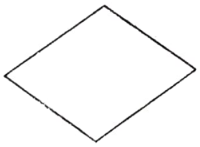


A magnetic disk

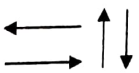
Input entry or output display operation.



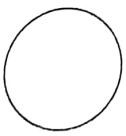
Connects remote portions of the flowchart not on the same page



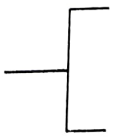
A decision making and branching operation that has two alternatives



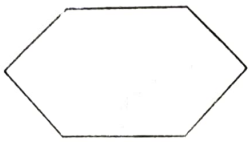
Flow lines



Connects remote parts of the flowchart on the same page



Add comments or furnish clarifications



Display

Algorithms –

(1) Write an algorithm for finding sum of the two numbers.

(2) Write an algorithm for finding remainder of a division operation where dividend and divisor and divisor are both integers.

(3) Write an algorithm to determine whether the accepted number is even or odd.

(4) Write an algorithm to exchange the values of two variables using the third variable.

(5) Write an algorithm to exchange the values of two variables without using the third variable.

(6) Write an algorithm to find out the highest out of two numbers.

(7) Write an algorithm to find out the highest out of three numbers.

(8) Write an algorithm that will accept marks out of 100 and print the grade.

(9) Write an algorithm that will accept three sides of triangles and comments about the triangle and will also calculate and print area and perimeter of the triangle.

(10) Write an algorithm to find sum of 1 + 2 + 3 + ...+ N, where N is accepted from user

(11) Write an algorithm that will accept N no. of numbers and will find sum and average of these numbers.

(12) Write an algorithm to find out whether the accepted number is prime or not.

(13) Write an algorithm to find out the factorial of a number.

(14) Write an algorithm to print first N terms of the Fibonacci series.

(15) Write an algorithm to find out sum of the digits present in the given number N and print it.

(16) Write an algorithm $1 + x + x^2 + x^3$ ..... upto N terms.

(17) Write an algorithm $1 + x + x^2/2! + x^3/3! + x^4/4!$..... upto N terms.

(18) Write an algorithm to largest out of many numbers.

(19) Write an algorithm to find out whether the accepted three digit number is Armstrong number or not.

3. EXPRESSING ALGORITHMS – SEQUENCE
4. CONCEPT OF SCALAR DATA TYPES

3.1 Introduction –

- The algorithms are independent of the programming language. To implement the algorithms on any computer or any programmable machine, a suitable programming language is to be used.
- To express and to implement the algorithms using any programming language we will have to learn basic constructs of the programming language.
- In our current course, we will be using 'C' programming language and hence we will begin with the basic constructs of 'C' language.

3.2 Character set –

- The character set is nothing but all possible characters that can be used within the language 'C'. In the language 'C', all the characters that belong to standard **ASCII** character set are included. The **ASCII** stands for **American Standard Code** for **Information Interchange.** (Viva)
- Following are some of the characters present in ASCII character set.

Alphabets	→	A – Z		a – z				
Digits	→	0 – 9						
Some Other Characters	→	+	-	*	/	%	#	.
		{	}	[	]	(	)	&
		?	$	^	,	:	;	@

- The standard ASCII character set has got in all 128 characters. Every character has got its own unique ASCII code. These code values range from 0 to 127.
- For example, ASCII values of 'A' to 'Z' are 65 to 90 respectively. The ASCII values of 'a' to 'z' are 97 to 122 respectively. The ASCII values of '0' to '9' are 48 to 57 respectively.

3.3 Keywords (Reserved Words) –

- The keywords are nothing but the words which has got predefined meaning in that language. As a programmer we can not change meaning of these keywords. These keywords must be used for their predefined purpose. The keywords are also called as reserved words.
- Following are keywords used in 'C' language.

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

3.4 Identifiers –

- An **identifier** is nothing but the name given to the programming element such as constant, variable, array, function etc.
- The purpose of identifiers is to identify these programming elements with their own names. The rules for forming identifier names are given as follows.
- Rules for identifier Name –

(1) An identifier name is to be made up of alphabets (letters) and digits. No other special characters are allowed.

(2) An identifier name can not start with digit or other words identifier must start with a letter.

(3) The underscore (_) is the only special character that is allowed to form the name of an identifier.

(4) The underscore is treated as an alphabet.

(5) The lowercase letters and the uppercase letter will differ from each other while forming the identifier names. For e.g. **a** and **A** are different identifier names. Similarly all the following eight identifier names are different from each other.

mnp mnP mNp mNP Mnp MnP MNp MNP

(6) The keywords are not allowed to be used as identifier names.

Following are some valid identifier names.

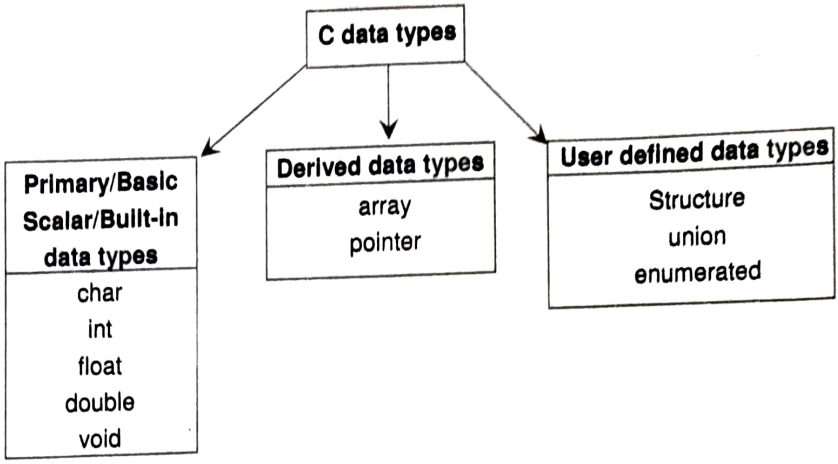
a abc abc123 a1b2c3 max_tamp _low_weight

Following are some invalid identifier names with the reasons of invalidity.

hello+xyz	the character '+' is not allowed
2abc	the identifier name must not begin with digit
hello welcome	the blank space is not allowed
while	the keyword is not allowed as identifier name.

3.5 DATA TYPES –

- The data types used in C can be classified as follows.



Integer data type (**int**) –

- The data type **Int** includes integer constants, integer variables and integer expressions.
- The integer number is nothing but sequence of digits which may be preceded by a plus(+) or by minus(–) sign.
- The decimal point or exponent sign or any other special character is not allowed.

Character data type (**char**) –

- The data type **char** includes character constants, character variables and character expressions.
- The character data type is made up of all the valid ASCII characters. There are 128 characters in the standard ASCII character set. Every character has got its own unique value between 0 to 127.

Float data type (**float**) –

- The data type **float** includes floating point constants, floating point variables and floating point expression.
- The floating point number is made up of digits 0 to 9 along with either a decimal point or exponent sign or both.
- The floating point number may be preceded with a plus(+) sign minus(-) sign. No other special characters are allowed.

Double data type (**double**) –

- The data type **double** includes double type constants, types variables and double type expressions.
- The double type number is also made up of digits 0 to 9 along either a decimal point or exponent sign or both.
- The double type number may be preceded with a plus(+) sign or minus(-) sign. No other special characters are allowed.
- Thus, the details of double type data are same as that of the float type data. The difference between the two is that the double type data will require double memory space for storage than that of the float type data.

Note – **Derived data types** and **User Defined data types** will be discussed in the further text as and on when suitable.

3.6 Data Types with Qualifiers –

- In 'C' there are four different qualifiers names as **signed, unsigned, short** and **long.** These qualifiers are used with the data types named as **int, float, double** and **char.** Following table shows all the valid combination of data types and qualifiers supported by C++.

Data Type With Qualifiers	Range of Values	Memory Bytes
char	– 128 to +127 (-2^7 to $2^7 - 1$)	1
signed char	– 128 to + 127 (-2^7 to $2^7 - 1$)	1
unsigned char	0 to 255 (0 to $2^8 - 1$)	1
short int	– 32768 to +32767 (-2^{15} to $2^{15} - 1$)	2
signed short int	– 32768 to +32767 (-2^{15} to $2^{15} - 1$)	2
unsigned short int	0 to 65535 (0 to $2^{16} - 1$)	2
int	– 32768 to +32767 (-2^{15} to $2^{15} - 1$)	2
signed int	– 32768 to +32767 (-2^{15} to $2^{15} - 1$)	2
unsigned int	0 to 65535 (0 to $2^{16} - 1$)	2
long int	–2147483648 to +2147483647 (-2^{31} to $2^{31} - 1$)	4
signed long int	–2147483648 to +2147483647 (-2^{31} to $2^{31} - 1$)	4
unsigned long int	0 to 4294967295 (0 to $2^{32} - 1$)	4
float	$\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{+38}$	4
double	$\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{+308}$	8
long double	$\pm 3.4 \times 10^{-4932}$ to $\pm 1.1 \times 10^{+4932}$	10

- The data type **char** requires one memory byte for storage. One memory byte is nothing but group of 8 bits. The **8-bit** binary number will result into 2^8 **i.e. 256** combinations. Half of the combinations are utilised for negative values and the remaining half of the combinations are utilised for positive values along with zero. Hence the range of values will be **-128 to 127.**

- The **signed char** has got same details as that of the **char**. This is because **char** is nothing but **signed char, by default.** Thus, the qualifier **signed** becomes **redundant.** All the characters from the **ASCII** character set with values ranging from **0 to 127** will fit into the range of **char** and **signed char.**

- The **unsigned char** requires one memory byte for storage. Hence all the 2^8 **i.e. 256** combinations will be utilised for non negative values. Hence the range of values will be **0 to 255.**

- The **int** requires two memory bytes for storage. Two memory bytes i.e.**16-bit** binary number will result into 2^{16} **i.e. 65536** combinations. Half of the combination are utilised for negative values and the remaining half of the combinations are utilised for positive values along with zero. Hence the range of values will be **-32768 to 32767.**
- The **signed int** has got same details as that of the **int**. This is because **int** is nothing but **signed int, by default.**

- For **unsigned int,** all the 2^{16} **i.e. 65536** combinations are utilised for non-negative values. Hence the range of values is **0 to 65535.**

- The explanation for **short int, signed short int** and **unsigned short int** remains same as that of the **int, signed int** and **unsigned int** respectively. it indicates that the qualifier **short is redundant**.

- The **long int** requires four memory bytes for storage. Four memory bytes i.e. **32-bit** binary number will result into 2^{32} combinations. Half of the combinations are utilised for negative values and the remaining half of the combinations are utilised for positive values along with zero. Hence the range of values represented by **long int** is given as - **2147483648 to 2147483647**.

- The **signed long int** has got same details as that of the **long int.** This is because **long int** is nothing but **signed long int, by default.**

- For **unsigned long int,** all the 2^{32} combinations are utilised for non-negative values. Hence the range of values given as **0 to 4294967295.**

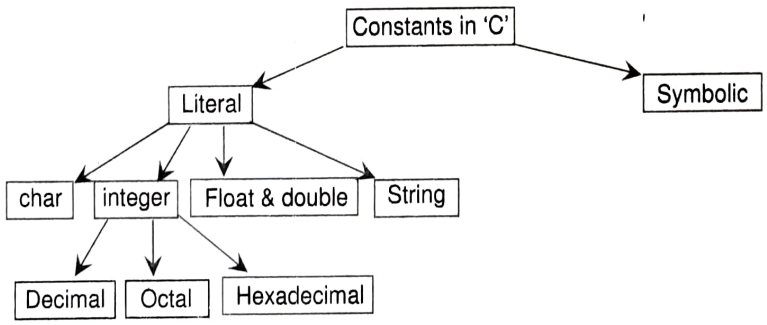
- The **float, double** and **long double** requires 4, 8 and 10 memory bytes for storage respectively. Range of values, for all the three is determined by **IEEE-754** format (Institute of Electrical and Electronics Engineers)

- The sizes in number of bytes for short int, int and long int are different for different machines as follow.

	8-bit machine	**16-bit machine**	**32-bit machine**	**64-bit machine**
short int	1 byte	2 bytes	2 bytes	2 bytes
int	2 bytes	2 bytes	4 bytes	4 bytes
long int	4 bytes	4 bytes	4 bytes	8 bytes

3.7 Constants –

- The constant is an entity whose value will not change throughout the execution of the program. The constant in 'C' are classified as follows.



3.7.1 Literal Constants –

Character constants –

The character constants are nothing but all the valid ASCII characters. The character constant when used in a program will be enclosed into single quotes i.e. single apostrophes.

For e.g.

Alphabets	→	'A'.....'Z'		'a'.....'z'				
Digits	→	'0'.....'9'						
Some Other Characters	→	'+'	'-'	'*'	'/'	'%'	'#'	'@'
		'{'	'}'	'['	']'	'('	')'	'&'
		'?'	'$'	'^'	','	':'	';'	'.'

The character constants also include all **backslash characters** which are also called as **escape sequences.** These character constants are given as follows

'\n'	new line (i.e. line feed) character	**'\f'**	form feed character
'\0'	null character	'\\'	backslash character
'\t'	tab character	'\''	single quote character
'\b'	backspace character	'\"'	double quote character
'\r'	carriage return character	**'\a'**	attention (beep or ring) character

Decimal Integer Constants –

- The decimal integer constants are integer number represented in decimal number system.
- These constants are sequence of digits from **0 to 9** which may be preceded by a plus(+) sign or minus(-) sign.
- These constants may end up with a letter **U or u** to indicate unsigned value. These constants may end up with a letter **L or** ℓ to indicate long value. These constants may end up with a letter **UL or u** ℓ to indicate unsigned long value.
- No other special characters are allowed. The decimal point or exponent sign are also not allowed.

Valid decimal integer constants –

0 +0 –0 123 +75 -2184 +71383 75000L

Invalid decimal integer constants with the reasons of invalidity –

3,128 → Comma is not allowed

7.0 → Even if the value of this number is 7 i.e. integer, the way the number is represented, is not the valid decimal integer constant. This is because, the decimal point is not allowed

7e+02 → The value of this number is 7 X 10^2 i.e. 700 which is an integer. However, the way the number is represented, is not the valid decimal integer constant. This is because, the exponent sign i.e. **e** is not allowed.

Octal Integer Constants –

- The octal integer constants are integer numbers represented in octal number system.

- These constants are sequence of digits from **0 to 7** which are not preceded by any sign.
- These constants are preceded by zero(0).
- These constants may end up with a letter **U or u** to indicate unsigned value. Theses constants may end up with a letter **L or ℓ** to indicate long value. These constants may end up with a letter **UL to uℓ** to indicate unsigned long value.
- No other special characters are allowed. The decimal point or exponent sign are also not allowed.

Valid octal integer constants –

0 0723 0234 0177777 017777777L

Invalid octal integer constants with the reasons of invalidity –

03,127 → Comma is not allowed.

078 → The digit 8 is not allowed.

345 → Invalid octal constant as the number is not preceded with zero(0). However, the compiler will not cause any error. This is because the compiler will treat this number as valid decimal integer constant.

07.3e+02 → The decimal point and exponent sign are not allowed.

+0123 → The plus(+) sign is not allowed.

–0123 → The minus(-) sign is not allowed

All the above examples except 345 will result into the compilation error.

Hexadecimal Integer Constants –

- The hexadecimal integer constants are integer number represented in hexadecimal number system.
- These constants are sequence of digits from **0 to 9** and letter from **A to F** or **a to f**. These constant are not preceded by any sign.
- These constants are preceded by **0x or 0X.**
- These constants may end up with a letter **U or u** to indicate unsigned value. These constants may end up with a letter **L or ℓ** to indicate long value. These constants may end with a letter **UL or uℓ** to indicate unsigned long value.
- No other special characters are allowed. The decimal point or exponent sign are also not allowed.

Valid hexadecimal integer constants –

0x 0x723 0x7AB 0X3fff 0xAbCd 0X7Ac2 0x7FFFFFL

Invalid hexadecimal integer constants with the reasons of invalidity –

0x3,128 → Comma is not allowed.

0x7AFH → The letter H is not allowed

345 → Invalid hexadecimal constants as the number is not preceded with 0x or 0X. However, the compiler will not cause any error. This is because, the compiler will treat this number as valid decimal integer constant.

A3fE → Invalid hexadecimal constant as the number is not preceded with 0x or 0X. The compiler will cause an error. This is because, the compiler will treat this number as decimal integer constant and it is not a valid decimal integer constant.

0x7.3e+02 → The decimal point and exponent sign are not allowed.

+0Xa2c3 → The plus(+) sign is not allowed.

-0x7fff → The minus(-) sign is not allowed.

All the above examples except 345 will result into the compilation error.

Float and Double type Constants –

- The float and double type constants are the numbers made up of digits 0 to 9 along with either a decimal point or exponent sign or both.
- No other special characters are allowed.
- The float and double type constants may get preceded by plus(+) or minus(-) sign.

Valid float and double constants –

0.0 -12.3 721.8 +19.3724 7.0 7e+02 -2.7132E+02 +3.1423e-04

Even if the value of 7.0 is 7 i.e. integer, the number 7.0 is a valid floating point constant. This is because it uses decimal point.

Even if the value of 7e+02 is 700 i.e. integer, the number 7e+02 is a valid floating point constant. This is because it used exponent sign.

Invalid float and double constants –

.7 → The decimal point must be present between the two digits.

7. → The decimal point must be present between the two digits.

7.e+02 → The decimal point must be present between the two digits.

1,234.56 → The comma is not allowed

- The above description, along with the examples is application to float type as well as double type constants.
- The difference between the two is that float type constant takes 4 memory bytes and the double type constants take 8 memory bytes for storage.
- The other difference is the range of values.

The range of float type constants is, $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{+38}$.

The range of double type constants is, $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{+308}$.

String Constants –

- The string constant is nothing but sequence of valid ASCII characters enclosed within double quotes i.e. double apostrophes (" ").
- The string constants are automatically terminated with a special character called as null character ('\0'). When the string constant is stored in memory then this character becomes useful to indicate the end of the string.

Valid string constants –

"Hello 123, Welcome" "A/123, xyz, abc, def, Dombivli" "C appears simple"
"C7&$qwer#@!" "\" Hi \" , Welcome" "ABCDE1234J"

Invalid string constants with the reasons of invalidity –

Hello → The opening and closing double quotes are missing
Welcome" → The opening double quote is missing.
"Hi → The closing double quote is missing.

Note – The **'A'** and **"A"** are different from each other. The **'A'** is a character constant and it requires one memory byte for storage. The **"A"** is a string constant and it requires two memory bytes for storage, one for the **'A'** and the other for the null character i.e. **'\0'**.

3.7.2 Symbolic Constants –

- In C, the symbolic constants can be obtained with three different methods.

(a) Using the keyword called as const.
(b) Using enumerated data types.
(c) Using pre-processor directives.

(a) Using the keyword called as const –

- The symbolic constants can be declared using the keyword called as **const.**
- Once these symbolic constants are declared in the program, then their values can not be changed but their values can be used wherever required in the program.
- When we use symbolic constant in a program then it will be replaced by equivalent value.
- Consider the following example.

```
const int X = 100;
const float PI = 3.14;
... ... ... ...
... ... ... ...
y = X + 25;
... ... ... ...
area = PI * r * r;
circum = 2 * PI * r;
... ... ... ...
X = 200;          // Results into compilation error
PI = 4.0;         // Results into compilation error
```

- In the above example, we have declared a symbolic constant called as X of type **int** whose equivalent value is 100 and we have declared as symbolic constant called as PI of type **float** whose equivalent value is 3.14.

- In the further three statements the values of these symbolic constants are utilised. It means that X and PI will be replaced by 100 and 3.14 respectively.

- In the last two statements we are making an attempt of changing the values of symbolic constants, which is not allowed and hence result into compilation error.

- **Advantage of declaring a symbolic constant can be given as** – If we use symbol PI many times in the program and later on we decide to change the equivalent value from **3.14** to **3.14159**, then just a one change where we have declared PI, is good enough and at all the place wherever we have used PI, it gets automatically replaced by 3.14159 instead of 3.14. If we would have used the value 3.14 directly then at so many places in the program we would have to replace the value 3.14 by the value 3.14159.

(b) Using enumerated data types –

- The symbolic constants can be declared with the help of enumerated data types. The enumerated data type is one of the user defined data types and are useful for writing self documenting codes.

- Consider the following examples.

(a) **enum direction {north, east, west, south};**

In the above example, **direction** is the enumerated data type with its values as **north, east west** and **south** whose internal values are 0, 1, 2 and 3 respectively.

The net effect is we have declared four symbolic constants named as north, east, west and south with their equivalent values 0, 1, 2 and 3 respectively.

(b) **enum weekdays {sun, mon, tue, wed, thu, fri, sat};**

In the above example, **weekdays** is the enumerated data type with its values as **sun, mon, tue, wed, thu, fri** and **sat** whose internal values are 0, 1, 2, 3, 4, 5 and 6 respectively

The net effect is we have declared seven symbolic constants named as sun, mon, tue, wed, thu, fri and sat with their equivalent values 0, 1, 2, 3, 4, 5 and 6 respectively.

(c) **enum cards {spade, heart, club, diamond};**

In the above example, **cards** is the enumerated data type with its values as **spade, heart, club** and **diamond** whose internal values are 0,1, 2 and 3 respectively.

The net effect is we have declared four symbolic constants named as spade, heart, club and diamond with their equivalent values 0, 1, 2 and 3 respectively.

(d) **enum colors {red, blue=10, pink, green, orange=30, magenta}**

In the above example, **colors** is the enumerated data type with its values as **red, blue, pink, green, orange** and **magenta** whose internal values are 0, 10, 11, 12, 30 and 31 respectively.

The net effect is we have declared six symbolic constants named as red, blue, pink, green, orange and magenta with their equivalent values 0, 10, 11, 12, 30 and 31 respectively.

Advantages of using enumerated data types –

(1) The enumerated data types are useful for writing the self documenting codes. For example, in case of a solitaire program, it is always better to use the words, **spade, heart, club** and **diamond** instead of using the values **0, 1,** 2 and **3** respectively.

(2) Multiple symbolic constants can be defined in a single line declaration.

Disadvantages of using enumerated data types –

(1) We can not read or print the values of enumerated data types.

(2) The values associated with the enumerated data types can not be floating point values.

(c) Using pre-processor directives –

- The symbolic constant can also be defined with the help of a pre-processor directive named as **#define.**
- Consider the following example.

 # define PI 3.14159

The effect of the above pre-processor directive is that we are defining constant called as PI whose equivalent value is 3.14159. Later on in the program, wherever the PI is used it will be replaced by its equivalent value i.e. 3.14159.

3.8 Variables –

- The variable is an entity whose value can change during the execution of the program. All the rules of identifier names are applicable to variable names.

Declaration of variable names –

The variable that we want to use in a program must be declared. The **general syntax** of declaring variable is as follows.

datatype variable name

or

datatype variable name1, variable name2, ..., variable name n

e.g. 1 **int x;**

In the above statement, we are declaring a variable **x** of type **int**. In response to this declaration, the compiler will allocate i.e. reserve a memory space of **two** bytes and it will name that space as **x,** so that later we can use that **x** in the program.

e.g. 2 **int p, c, m, total;**

In the above statement, we are declaring four variables **p, c, m** and **total** of type **int**. In response to this declaration, the complier will allocate (reserve) a memory space of **two** bytes **each** and it will name that space with respective variable name. Once these variables are declared, later we can use them in the program.

e.g. 3 **int x;**

float y;

char z;

In the above statements, we are declaring three variables **x, y** and **z** of type **int, float** and **char** respectively. In response to these declarations, the compiler will allocate i.e. reserve a memory space of **two** bytes for **x, four** bytes for **y** and **one** byte for **z.**

Note – The variables must be declared before they are used.

Initialisation while declaration –

- Consider the following example.

  Int count;

- Due to this declaration, the complier will allocate a memory space of two bytes and it will name that space as count. However the value of this space is not guaranteed. It can be anywhere between –32768 to +32767.

  Hence it is said that the value of the count will be **garbage.**

- Suppose in our application we wan that initial value of count to be zero and not the garbage then we must initialize the value of count to zero at the time of declaration only. It can be easily obtained as follows.

  int count = 0;

- The initial value need not be always zero. It can be any valid value according to the application. For e.g.

  int x = 25;

  float y = 4.5;

  char z = 'Q';

3.9 Operators –

All the operators used in C++ are given as follows with their precedence and associativity.

Operators	Associativity
() [] → . postfix++ postfix– –	Left to Right
→ prefix++ prefix– – ! ~ unary+ unary– unary* unary& (type) sizeof	**Right to Left**
* / %	Left to Right
+ –	Left to Right
<< >>	Left to Right
< <= > >=	Left to Right
== !=	Left to Right
&	Left to Right
^	Left to Right
|	Left to Right
&&	Left to Right
||	Left to Right
?:	Left to Right
= *= /= %= += –= <<= >>= &= ^= |=	**Right to Left**
,	Left to Right

Mathematical Operators –

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus operator to give remainder after division

Examples –

16 % 5 → 1 10 % 5 → 0 13 % 7 → 6 5 % 7 → 5

-13 % 5 → -3 -13 % - 5 → -3 13 % - 5 → 3 13 % 5 → 3

Relational Operators –

Operator	Purpose
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical operators –

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

The explanation of Logical operators can be given as follows.

A	B	A && B	A || B	!A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Assignment Operator –

- The assignment operator is used to form assignment statement.
- The **general syntax** of the assignment statement is given as follows.

variable = expression

The expression at the right hand side is solved first to result into a single value and that value will be assigned to the variable present at the left hand side. The previous value of the variable will be replaced by a new value.

e.g.1

```
int x, y, z;
.....
.....
x = y + z;
```

The values of y & z will be added and the answer of addition will be assigned to the variable x. The previous value of x will be overwritten by the newly assigned value. The values of y and z will remain unchanged.

e.g.2

```
int count;
.....
.....
count = count + 1;
```

Due to the above assignment statement, 1 will be added with the value of the count and the answer of addition will be assigned back to the variable count. The net effect is, the value of count is incremented by 1.

Bitwise Operators –

Operator	Purpose
&	Bitwise AND
|	Bitwise OR
^	Bitwise EX-OR
~	Bitwise NOT (one's complements)
<<	Bitwise Left Shift
>>	Bitwise Right Shift

To understand the bitwise operators, consider the following examples.

Int x = 19, y = 26, z;

1) z = x & y;

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$y = 26_{10}$
0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	0	$z = x\&y = 18_{10}$

2) z = x | y;

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$y = 26_{10}$
0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	1	$z = x \mid y = 27_{10}$

3) z = x ^ y;

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$y = 26_{10}$
0	0	0	0		0	0	0	0		0	0	0	0		1	0	0	1	$z = x \wedge y = 9_{10}$

4) z = ~ x; z = ~ x = -20_{10} if treated as signed or 65516_{10} if treated as unsigned

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
1	1	1	1		1	1	1	1		1	1	1	0		1	1	0	0	z = ~x

For details Refer Notebook

5) z = ~ y; z = ~ y = -27_{10} if treated as signed or 65509_{10} if treated as unsigned

0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$x = 26_{10}$
1	1	1	1		1	1	1	1		1	1	1	0		0	1	0	1	z = ~y

6) z = x << 1;

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
0	0	0	0		0	0	0	0		0	0	1	0		0	1	1	0	$z = x<<1 = 38_{10}$

7) z = x << 3;

0	0	0	0		0	0	0	0		0	0	0	1		0	0	1	1	$x = 19_{10}$
0	0	0	0		0	0	0	0		1	0	0	1		1	0	0	0	$z = x<<3 = 152_{10}$

8) z = y >> 1;

0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$x = 26_{10}$
0	0	0	0		0	0	0	0		0	0	0	0		1	1	0	1	$z = y<<1 = 13_{10}$

9) z = y >> 3;

0	0	0	0		0	0	0	0		0	0	0	1		1	0	1	0	$x = 26_{10}$
0	0	0	0		0	0	0	0		0	0	0	0		0	0	1	1	$z = y<< 3 = 3_{10}$

Note 1 – If the binary number is shifted to left by 'n' number of positions, then its value gets multiplied by 2^n

Note 2 – If the binary number is shifted to right by 'n' number of positions, then its value gets divided by 2^n.

Updating Assignment Operators –

The **general syntax** of the updating assignment statement is given as follow.

variable op = expression

The expression at the right hand side is solved first to result into a single value and the variable at the left hand side is updated with the value. The type of the update depends on the **op.**

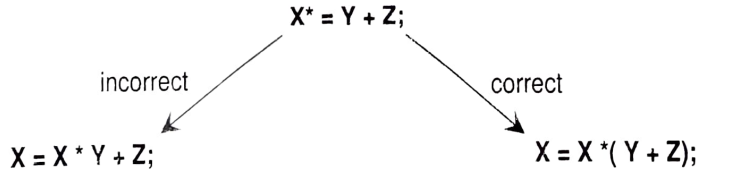
Following are the different updating assignment operators used in C++.

+=	-=	*=	/=	%=
&=	|=	^=	<<=	>>=

- The effect of the above operators can be understood from the following examples.

Operation	Equivalent Operation
X += 1;	X = X + 1;
Y -= 2;	Y = Y – 2;
Z *= 3;	Z = Z * 3;
W /= 4;	W = W / 4;
U %= 5;	U = U % 5;
A &= B;	A = A & B;
A |= B;	A = A | B;
A ^= B;	A = A ^ B;
A <<= 2;	A = A << 2;
B >>= 3;	B = B >> 3;

- Consider the following example.



- In the above example, the statement using the updating assignment operation is shown, and to understand its effect two possible options are shown. The option at the left hand side is incorrect whereas the option shown at right is correct. The **justification** of correct option is given as follows.

- **Justification** – When we use any Updating Assignment Operator then the expression at the right hand side is solved first to result into a single value. The variable at the left hand side is then updated with this value. The type of the update is decided by the basic operation. Hence, Y + Z is to be evaluated first and then with the answer of their addition, the value of X is to be updated. The update is multiplicative. Thus, addition is to be done and then the multiplication.

Increment Operator (++)

- The increment operator (++) is used increment the value of a variable by 1.
- The increment operator is either used as pre-increment operator or post-increment operator.

Consider the following example.

Pre-Increment	Post-Increment
x = 3; **++ x;**	**x = 3;** **x ++;**
Due to pre-increment operator the value of x will be incremented by 1 and it will become 4.	Due to post-increment operator the value of x will be incremented by 1 and it will become 4.

From the above example, it seems like there is no difference between pre-increment and post-increment operator. However, that is not the reality. To understand the difference between the two, consider the following example.

Pre-increment	Post-Increment
X = 3; **y = ++ x;**	**x = 3;** **y = x ++;**
In the above statement there are two operations, one is increment and the other is assignment.	In the above statement there are two operations, one is increment and the other is assignment.
Since the increment operator is used as **pre-increment** operator, the value of x will be incremented first, so that the value of **x** will become **4.**	Since the increment operator is used as **post-increment** operator, the assignment will be done first and increment will be done later. Hence the existing value of x i.e. 3 is assigned to y and hence the value of **y** will become **3**.
Then the incremented value of x i.e. 4 is assigned to y and hence value of **y** will become **4**. Thus, the assignment statement, y = ++ x; will be equivalent to, **x = x + 1;** **y = x;**	Then the value of x will be incremented by 1 and **x** will become **4**. Thus, the assignment statement, y = x ++ ; will be equivalent to, **y = x;** **x = x + 1;**

Decrement Operator (– –)

- The decrement operator (– –) is used to decrement the value of a variable by 1.
- The decrement operator is either used as pre-decrement operator or post-decrement operator.

Consider the following example.

Pre-decrement	Post-decrement
x = 3; **– –x;** Due to pre-decrement operator the value x will be decrement by 1 and it will become 2.	**x = 3;** **x – –;** Due to post-decrement operator the value of x will be decremented by 1 and it will become 2.

From the above example, it seems like there is no difference between pre-decrement and post-decrement operator. However, that is not the reality. To understand the difference between the two, consider the following example.

Pre-decrement	Post-decrement
x = 3; **y = – –x;** In the above statement there are two operations, one is decrement and the other is assignment. Since the decrement operator is used as **pre-decrement** operator, the value of x will decrement first, so that the value of **x** will become **2**. Then the decremented value of x i.e. 2 is assigned to y and hence value of **y** will become **2**. Thus, the assignment statement, y = – – x; will be equivalent to, **x = x – 1;** **y = x;**	**x = 3;** **y = x – –;** In the above statement there are two operations, one is decrement and the other is assignment. Since the decrement operator is used as **post-decrement** operator, the assignment will be done first and decrement will be done later. Hence the existing value of x i.e. 3 is assigned to y and hence the value of **y** will become **3**. Then the value of x will decremented by 1 and **x** will become **2**. Thus, the assignment statement y = x – –; will be equivalent to **y = x;** **x = x – 1;**

Exercise –

In each of the following examples, determine the value of x and y.

x = 19; **y = ++x;** **∴ x =** 20 **y =** 20	**x = 19;** **y = x++;** **∴ x =** 20 **y =** 19
x = 19; **y = – – x;** **∴ x =** 18 **y =** 18	**x = 19;** **y = x– –;** **∴ x =** 18 **y =** 19

TypeCasting –

The type casting is the process of converting value of one type to other. The type casting is also called as coercion. Consider the following examples. This can be achieved as follows.

(data type) expression

Example–1:

```
int x=7;
float y;
y = (float)x;
```

In the above example, the type casting operation **(float)** will operate on the value of x and will type cast it to a float value i.e. 7.0 and hence the value that will be assigned to y will be 7.0. The x will still remain of type int with value 7.

Example–2:

```
int x = 65;
char z;
z = (char)x;
```

In the above example, the type casting operator **(char)** will operate on the value of x and will type cast it to a char value i.e. 'A' and hence the value that will be assigned to z will be 'A'. The x will still remain of type int with value 65.

Example–3:

```
float x = 10.8;
int y;
y = (int)x;
```

In the above example, the type casting operator **(int)** will operator on the value of x and will type cast it to int value i.e. 10 and hence the value that will be assigned to y will be 10. The x will still remain of type float with value 10.8. When the floating value is type casted to int value, then the number is truncated and not rounded i.e. the value 10.8 is changed to 10 and not 11.

Conditional Operator or Ternary Operator (?:) –

- The conditional operator (?:) is used to carry out simple conditional operations, The operator is also called as ternary operator because it is the only operator that works with three operands. The general form of using the conditional operator is as follow.

  expression 1 ? expression 2 : expression 3

- The expression1 is evaluated first which will be treated as logical expression.
- If expression1 result in any non-zero (true) value, then the expression2 is evaluated and its answer will be the final answer of the overall expression.
- If expression1 results in any zero (false) value, then the expression3 is evaluated and its answer will be the final answer of the overall expression.

- Consider the following example.

z = (y > 10)? a + 5 : a – 5

In the above example, the expression **y > 10** is evaluated first. If it results in value **true** then the expression, **a + 5** is solved and its answer is assigned to **z**. If the answer of the expression **y > 10** results in value **false** then the expression, **a – 5** is solved and its answer is assigned to **z.**

- Consider the following example.

min = (p < q) ? p : q;

In the above example, the expression **p < q** is evaluated first. If it results in value **true** then value of **p** is assigned to **min** and if it results in values **false** then the value of **q** is assigned to **min**.

Note –

Few more operators are yet to be discussed but those will be covered in the further text wherever appropriate. We will discuss three terms before we proceed to next topic.

Unary Operators – The operators that operate with a single operand are called as unary operator. Till the point we have finished four unary operators and those are:
!, ~, ++, – –.

Binary Operators – The operators that operate with two operands are called as binary operators. Except the above discussed four unary operators, all the operators that we have finished till the point are binary operators.

Ternary Operators– The operators that operate with three operands are called as ternary operators. There is only one ternary operator i.e. **?:**

3.10 Expressions –

- The expressions are formed with the help of **operands** and **operators.**
- The two broad categories of the expressions are given as follows.
  (a) Arithmetic (Mathematical) expressions.
  (b) Boolean (Logical) expression.
- The Arithmetic (Mathematical) expression will result into a value which will be any valid integer number or float type number or double type number.
- The Boolean (logical) expression can result only into two values, either true or false.

Exercise –

Write the expressions for the following statements.

(1) a is equal to 30 and b is not equal to 10.

Ans:

(2) Write the expression whose value will be true with the following conditions.

A student must clear MECHANICS and should clear at least one subject out of BEE and CP.

Ans:

(3) The product of y and z is added with the value of x. From that answer of addition, the w is being subtracted and the net answer is assigned to R.

Ans:

(4) Write the expression for the following.

$$y = 1 + \cfrac{x}{1 + \cfrac{x}{1 + \cfrac{x}{1 + x}}}$$

Ans:

(5) Write the expression for the following.

$$x = \frac{-b + (b * b) + 2 - 4ac}{2a}$$

Ans:

(6) Write the expression for the following.

$$x = \frac{2v + 6.22(c + d)}{g + v}$$

Ans:

3.11 Pre–Processor Directives –

- The pre-processor directives are the **directions** given to the compiler to carry out specific task even before the compilation of the program starts.
- Since we are **directing** the complier to carry out specific task **before** the compilation **process** starts it is referred as **pre-processor directives.**
- The pre-processor directives are not the statements of the language. These directives starts with #.
- The pre-processor directives never end with semicolon.
- Following are the commonly used pre-processor directives.

(3) The product of y and z is added with the value of x. From that answer of addition, the w is being subtracted and the net answer is assigned to R.

Ans:

(4) Write the expression for the following.

$$y = 1 + \cfrac{x}{1 + \cfrac{x}{1 + \cfrac{x}{1 + x}}}$$

Ans:

(5) Write the expression for the following.

$$x = \frac{-b + (b * b) + 2 - 4ac}{2a}$$

Ans:

(6) Write the expression for the following.

$$x = \frac{2v + 6.22(c + d)}{g + v}$$

Ans:

3.11 Pre–Processor Directives –

- The pre-processor directives are the **directions** given to the complier (compiler) to carry out specific task even before the compilation of the program starts.
- Since we are **directing** the complier to carry out specific task **before** the compilation **process** starts it is referred as **pre-processor directives.**
- The pre-processor directives are not the statements of the language. These directives starts with #.
- The pre-processor directives never end with semicolon.
- Following are the commonly used pre-processor directives.

(a) **# define –**

The pre-processor **# define** is used to declare the symbolic constants.

Consider the following example.

 # define PI 3.14159

In the above example, we define a symbolic constant named as PI whose equivalent value will be 3.14159. With this directive we have informed the complier that the PI is a symbolic constant with its own value. Once the compilation of the program starts, wherever the PI occurs in the program, it will be replaced by its equivalent value.

(b) **#include –**

The pre-processor directive **# include** is used to include specific files within a program.

There are many facilities available in the core of 'C' language. However, there are many more facilities which are available outside of the core of 'C'. These facilities are available in specific files. These files are called as **header files** and are also called as **library files.**

Consider the following example.

(1) To use the mathematical facilities such as **sin( ), cos( ), tan( ), log( ), sqrt( )** etc. a file called as **math.h** must be included. The corresponding pre-processor directive will be given as follows

#include<math.h>

Due to the above pre-processor directive, we inform i.e. we **direct** the compiler to include a file called as **math.h**, so that if we use any of the above discussed facilities, the compilation error will not occur and the facilities will become accessible.

(2) The **scanf( )** and **printf( )** are the functions for reading and displaying the data. To use these functions, we have to include a file called as **stdio.h**. The corresponding pre-processor directive will be given as follows

#include<stdio.h>

The stdio.h refers to the standard input output header file. Once we include the file **stdio.h**, we can use scanf( ) and printf( ) in our programs.

(3) The **clrscr()** is a function used to clear the screen i.e. the display area. The **getch()** is a function used to get a character from the input device i.e. keyword. To use these functions we have to include a file called as **conio.h**. The corresponding pre-processor directive will be given as follows.

#include<conio.h>

The conio.h refers to the console input output header file. Once we include the file **conio.h**, we can use clrscr( ) and getch() in our programs.

There are few more pre-processor directives used in 'C'. Those are listed as follows.

#if #else #elif #endif #ifdef #ifndef

3.12 Data Input and Output –

- The three common steps in writing the programs are,
  (a) Accept the data from the user, entered from standard input device. (**data input**)
  (b) Process the input data to produce the required result. (**data processing**)
  (c) Display the results for the user on the standard output device. (**data output**)
- The general purpose data input facility is nothing but a function **scanf( )**.
- The general purpose data output facility is nothing but a function **printf( )**.

scanf( ) –

- The scanf( ) is a function which is used to read the data items from the standard input device.
- The scanf( ) function is defined within a file **stdio.h** and hence we must include this file within a program.
- The general format of the function scanf( ) is as follows.

  scanf(control string, arg1, arg2,.....,argn)

  The control string refers to the string containing certain required formatting information and the arguments are the addresses of the variables in which the data is to stored.
- The control string consists of groups of characters. Every group is made up of percent sign(%) followed by a conversion specification character. This character indicates the type of input data item.

  e.g.

  scanf("%d",&x);

  The above example used a conversion specification character **d**. It means that the input integer data will be converted to decimal format and it will be stored at **address-of** x. In other words the accepted number is converted to decimal format and it is stored into x.
- There are number of conversion specification characters which are used with input data. These characters are listed below.

Conversion Specification Character	Meaning
c	Input data is a single character.
d	Input data is decimal integer.
e	Input data is a floating point number.
f	Input data is floating point number.
g	Input data is a floating point number.
i	Input data is either decimal or octal or hexadecimal integer.
o	Input data is a octal integer.
s	Input data is a string which will be accepted till the white space.
u	Input data is a unsigned decimal integer.
x	Input data is a hexadecimal integer.
[...]	Input data is a string with white space characters.

Consider the following example.

```
int x;
float y;
char z;
scanf("%d %f %c", &x, &y, &z);
```

The above scanf( ) will read first data as decimal integer and will store it at address of x, will read second data as floating point number and will store it at address of y, will read third data as a character and will store it at address of z.

printf( ) –

- The printf( ) is a function which is used to write the data items to the standard output device.
- The printf( ) function is defined within a file **stdio.h** and hence we must include this file within a program.
- The general format of the function printf( ) is as follows.

  printf(control string, arg1, arg2,.....,argn)

  The control string refers to the string containing certain required formatting information and the arguments can be constants, variables or expressions.
- The control string consists of group of characters. Every group is made up of a percent sign(%) following by a conversion specification character. This character indicates the type of output data item.

  e.g.

  printf("%d", x);

  The above example uses a conversion specification character **d**. It means that the value of **x** will be converted to decimal format and it will be printed.
- There are number of conversion specification characters which are used with output data. These characters are listed below.

Conversion Specification Character	Meaning
c	Data displayed is a single character.
d	Data displayed is a decimal integer.
e	Data displayed is a floating point number with exponent.
f	Data displayed is a floating point number without exponent.
g	Data displayed is a floating point number either using e-type or f-type conversion. Trailing zeroes, trailing decimal point will not be displayed.
i	Data displayed is either decimal or octal or hexadecimal integer.
o	Data displayed is an octal integer without leading 0.
s	Data displayed is a string.
u	Data displayed is unsigned decimal integer.
x	Data displayed is a hexadecimal integer without leading 0x.

Consider the following example.

```
int x = 7;
float y = 2.2;
printf("%d %f", x, y);
```

The first number to be printed is of type decimal integer and the corresponding value to be printed is the value of variable x of type int. The second number to be printed is of type float and the corresponding value to be printed is the value of variable y of type float.

3.13 Use of standard functions / library functions –

- There are many facilities available in the core of 'C' language. However, there are many more facilities which are available outside of the core of 'C'. These facilities are developed as functions. These facilities are available in specific files. These files are called as **header files** and are also called as **library files**. The pre-processor directive **#include** is used to include these header/library files within a program.

- Following are some of the commonly required library functions.

Function	Return type	Description	Header file
abs(i)	int	Return absolute value of i.	stdlib.h
fabs(d)	double	Return absolute value of d.	math.h
labs(l)	long int	Return absolute value of l.	math.h
acos(d)	double	Return the arc cosine of d.	math.h
asin(d)	double	Return the arc sine of d.	math.h
atan(d)	double	Return the arc tangent of d.	math.h
atan2(d1, d2)	double	Return the arc tangent of d1/d2.	math.h
sin(d)	double	Return sine of d.	math.h
cos(d)	double	Return cosine of d.	math.h
tan(d)	double	Return tangent of d.	math.h
sinh(d)	double	Return hyperbolic sine of d	math.h
cosh(d)	double	Return hyperbolic cosine of d	math.h
tanh(d)	double	Return hyperbolic tangent of d	math.h
ceil(d)	double	Return a value rounded up to the next higher integer	math.h
floor(d)	double	Return a value rounded down to the next lower integer	math.h
exp(d)	double	Raise e to the power d. (e = 2.7182818…is the base of natural system of logarithms)	math.h
log(d)	double	Return the natural logarithm of d	math.h
log10(d)	double	Return the logarithm (base 10) of d	math.h
pow(d1, d2)	double	Return d1 raised to power d2	math.h
sqrt(d)	double	Return the square root of d.	math.h
fmod(d1, d2)	double	Return the remainder of d1/d2 (with same sign as d1)	math.h
rand( )	int	Returns a random positive integer.	stadlib.h
atof(s)	double	Convert string s to a double precision quantity	stdlib.h
atoi(s)	int	Convert string s to an integer	stdlib.h
atol(s)	long int	Convert string s to long integer	stdlib.h

isalpha(c)	int	Determine if argument is alphabetic. Return a non-zero value if true and return 0 otherwise.	ctype.h
isdigit(c)	int	Determine if argument is decimal digit. Return a non-zero value if true and return 0 otherwise.	ctype.h
isodigit(c)	int	Determine if argument is octal digit. Return a non-zero value if true and return 0 otherwise.	ctype.h
isxdigit(c)	int	Determine if argument is hexadecimal digit. Return a non-zero value if true and return 0 otherwise.	ctype.h
isalnum(c)	int	Determine if argument is alphanumeric. Return a non-zero value if true and return 0 otherwise.	ctype.h
islower(c)	int	Determine if argument is lowercase. Return a non-zero value if true and return 0 otherwise	ctype.h
isupper(c)	int	Determine if argument is uppercase. Return a non-zero value if true and return 0 otherwise.	ctype.h
ispunct(c)	int	Determine if argument is a punctuation character. Return a non-zero value if true and return 0 otherwise.	ctype.h
isspace(c)	int	Determine if argument is a white space character. Return a non-zero value if true and return 0 otherwise.	ctype.h
tolower(c)	int	If c is a uppercase letter then returns the corresponding lowercase letter else returns the original letter c.	ctype.h
toupper(c)	int	If c is a lowercase letter then return the corresponding uppercase letter else returns the original letter c.	ctype.h
getc(f)	int	Gets a single character from file f	stdio.h
getchar( )	int	Gets a single character from standard input device	stdio.h
gets( )	char*	Gets a single string as input from standard input device	stdio.h
putc(c, f)	int	Sends a single character c to file f.	stdio.h
putchar(c)	int	Sends a single character to standard output device	stdio.h
puts(s)	int	Sends a string s to standard output device	stdio.h
fopen(s1, s2)	file*	Opens a file named as s1 of type s2.	stdio.h
fclose(f)	int	Close file f. Return 0 if the file is closed successfully.	stdio.h
feof(f)	int	Determines whether end of file condition has occurred or not. If yes then returns non zero else returns zero.	stdio.h

fread(s, i1, i2, f)	int	Read i2 number of data items each of size i1 bytes from file f to string s	stdio.h
fwrite(s, i1, i2, f)	int	Writes i2 number of data items each of size i1 bytes from string s to file f.	stdio.h
fseek(f, l, i)	int	Move the pointer for file f to a distance of l bytes from the location i. The l might be beginning of the file or current position of file or ending of file.	stdio.h
ftell(f)	long int	Returns the current pointer position within the file f.	stdio.h
rewind(f)	void	Move the pointer to the beginning of file	stdio.h
fprintf(f,....)	int	Sends the data items to file f	stdio.h
fscanf(f,....)	int	Reads the data items from file f	stdio.h
printf(f.....)	int	Sends the data items to standard output device	stdio.h
scanf(f,.....)	int	Read the data items from standard input device.	stdio.h
strcmp(s1,s2)	int	Compare two strings lexicographically. Return negative value if s1 < s2, return 0 if s1 = s2 and return positive value if s1 > s2	string.h
strcmpi(s1,s2)	int	Compare two strings lexicographically by ignoring the case. Return negative value if s1 < s2, return 0 if s1 = s2 and return positive value if s1 > s2.	string.h
strcpy(s1,s2)	char*	Copy string s2 to s1.	string.h
strcat(s1,s2)	char*	Concatenate string s2 to s1. The resultant string is with s1. The s2 remains unchanged.	string.h
strlen(s)	int	Returns number of characters in string s.	string.h
strset(s,c)	char*	Set all the character within s (excluding terminating character\0) with c	string.h

Note –

c denotes a character type argument.

d denotes a double precision type argument.

f denotes a file argument

i denote integer type argument

l denote long integer type argument

s denote string type argument.

3.14 Programs –

CWP1– Write the output of the following program.

Ans – Refer your notebook for the program and the output.

Explanation –

- Every C program must have a function called as **main function**. The word **int** in the header of the main function indicates that the main function will return integer value to the operating system.
- The **{** is used to indicates the beginning of the main( ) and the **}** is used to indicate the ending of the main ( ).
- In the above program we are using printf( ), which is not available in the core of 'C' and hence to use printf( ) in the program we have to include a file called as **stdio.h** (standard input output header file).
- In the above program we are using clrscr( ) and getch( ), which are not available in the core of 'C' and hence to use clrscr( ) and getch( ) in the program we have to include a file called as **conio.h** (console input output header file).
- The string that is used as an argument of printf( ) is displayed on the output device. The '\n' is a special character that is used to take the cursor to the new line i.e. next line.

CWP2 – Write a program that will accept two integers and will print answers of their additions, subtraction, multiplication and division.

Method1 – Refer your notebook for the program and the output.

Method2 – Refer your notebook for the program and the output.

CWP3 – Write the output of the following program which will handle the typecasting.

Ans.– Refer your notebook for the program and the output.

Explanation –

1) **d1 = x/y;**

The value of x/y i.e. 13/5 **integer / integer will result into integer** and hence the value of d1 will be 2.

2) **d2 = x/y;**

Even if the type of d2 is float, the expression at the right hand side is still integer/integer, which will result into integer value i.e. 2. This value will be assigned to d2 and hence the value of d2 will become 2.0. When d2 will be printed using printf( ), the value printed will be 2.000000. This is because in 'C' by default six digits are printed after the decimal point.

3) **d3 = (float)(x/y);**

The answer of x/y i.e. 13/5 i.e. integer/ integer will result into integer i.e. 2. The type casting operator (float) will operate on this value and will type cast it to 2.0. Hence the value of d3 will be 2.0. When d3 will be printed using printf( ), the value printed will be 2.000000 for the same reason as discussed in point-2.

4) **d4 = (float)x/y;**

The type casting operator (float) will operate on the value of x i.e. 13 and will typecast it to 13.0 and it will be divide by 5. The 13.0 / 5 i.e. **float / integer will result into float value** i.e. 2.6. When d4 will be printed using printf( ), the value printed will be 2.600000 for the same reason as discussed in point-2.

5) d5 = x/(float)y;

The type casting operator (float) will operate on the value of y i.e. 5 and will typecast it to 5.0. The division 13 / 5.0 i.e. **int / float result into float value** i.e. 2.6. When d5 will be printed using printf( ), the value printed will be 2.600000 for the same reason as discussed in point-2.

6) d6 = (float)x/(float)y;

The type casting operator (float) will operate on the value of both x and y i.e. 13 and 5 and will type cast it to 13.0 and 5.0 respectively. The division i.e. 13.0 / 5.0 i.e. **float / float will result into float value** i.e. 2.6. When d6 will be printed using printf( ), the value printed will be 2.600000 for the same reason as discussed in point-2.

Note:

int / int	→	**int**
int / float	→	**float**
float / int	→	**float**
float / float	→	**float**

Formatted input –

- The consecutive non white space characters that form a data item define a field. It is possible to limit number of such characters by specifying a maximum field width for data item.
- To do this, the unsigned integer number indicating the field width is placed in the control string and it is placed between the % sign and the conversion specification character.
- The data item can be of fewer characters than then field width. If the number of characters exceed the specified field width, then any number of characters beyond the specified field width will not be read. These characters may be reads for the next data item.
- Let us consider the following programming example.

Test1.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int x,y,z;
      clrscr();
      printf("Enter data=");
      scanf("%3d%3d%3d", &x, &y, &z);

      printf("x=%d,y=%d,z=%d/n",x,y,z);
      getch();
      return 0;
}
```

After executing the given program,

- If the data entered is as follows,

  9 8 7

  Then following output will result,

  x=9 y=8 z=7

- If the data entered is as follows,
  987 65 432
  Then following output will result.
  x=987 y=65 z=432

- If the data entered is as follows,
  987654321
  Then following output will result.
  x=987 y=654 z=321

- If the data entered is as follows,
  9876 5432 1
  Then following output will result.
  x=987 y=6 z=543
  The two digits 2 and 1 are ignored if there is no further scanf( ) in the program, otherwise these two digits will be used by next scanf( ) statement.

- Let us consider one more programming example.

Test2.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int x;
     float y;
     char z;
     clrscr();
     printf("Enter data=");
     scanf("%4d %6f %c",&x,&y,&z);

     printf("x=%d,y=%f,z=%c\n",x,y,z);
     getch();
     return 0;
}
```

After executing the given program

- If the data entered is as follows,
  98 3.434567 E
  Then following output will result.
  x=98 y=3.434500 z=6
- If the data entered is as follows
  98765 E
  Then following output will result.
  x=9876 y=5.000000 z=E

- Let us consider one more programming example.

Test3.c

```
#include<stdio.h>
#include<conio.h>
int main()
{
      char x,y,z;

      clrscr();
      printf("Enter data=")';
      scanf("%c%c%c",&x,&y,&z);

      printf("x=%c,y=%c,z=%c\n",x,y,z);
      getch();
      return 0;
}
```

After executing the given program,

- If the data entered is as follows,
  abc
  Then following output will result.
  x=a y=b z=c
- If the data entered is as follows,
  a b c
  Then following output will result.
  x=a y=b z=c

Formatted Output –

- The data to be printed can be printed in different formats on the screen. This is done by specifying different field widths along with the conversion specification character. Let us consider following programs.

CWP4– White the output of the following program.

Ans– Refer your notebook for the program and the output.

Explanation –

- First group –The value of p i.e. 175 will be printed and the \n will take the cursor to the next line. Similarly, the value of q i.e. –175 will be printed and the \n will take the cursor to the next line.

- Second group–The %6d means that total width that will be used to print the decimal integer is of 6 columns. The number will be right justified in the given width. The value of p i.e. 175 will require 3 columns. Hence, 6 – 3 = 3 blank spaces are left out first and then the value 175 is printed. The value of q i.e. –175 will require 4 columns. Hence, 6 – 4 = 2 blank spaces are left out first and then the value –175 is printed.

- Third group –The %5d means that total width that will be used to print the decimal integer is of 5 columns. The number will be right justified in the given width. The value of p i.e. 175 will require 3 columns. Hence, 5 – 3 = 2 blank spaces are left out first and then the value 175 is printed. The value of q i.e. – 175 will require 4 columns. Hence, 5 – 4 = 1 blank space is left out first and then the value – 175 is printed.

- Fourth group– The %06d means that total width that will be used to print the decimal integer is of 6 columns. The Number will be right justified in the given width. While justifying the number to right, instead of leaving blank spaces those will be filled with required number of 0's(zeroes). The value of p i.e. 175 will require 3 columns. Hence, 6 – 3 = 3 zeroes are printed first and then the value 175 is printed. The value of q i.e. –175 will require 4 columns. Hence, 6 – 4 = 2 zeroes are printed before the magnitude 175 and the minus sign comes even before that.

- Fifth group –The %05d means that width that will be used to print the decimal integer is of 5 columns. The number will be right justified in the given width. While justifying the number to right, instead of leaving black spaces those will be filled with required number of 0's(zeroes). The value of p i.e. 175 will require 3 columns. Hence, 5 – 3 = 2 zeros are printed first and then the value 175 is printed. The value of q i.e. –175 will require 4 columns. Hence, 5 – 4 = 1 zero is printed before the magnitude 175 and the minus sign comes even before that.

- Sixth group –The %2d means that total width that will be used to print the decimal integer is of 2 columns. However, both the values of p and q i.e. 175 and –175 respectively will require more than 2 columns and hence the effect of %2d is ignored and the values 175 and – 175 respectively.

CWP5– Write the output of the following program.

Ans.– Refer your notebook for the program and the output.

Explanation –

- First Group – When the floating point number is printed using %f notation, then six digits are compulsorily printed after the decimal point. Hence, the value of p i.e. 12.7156 is printed as 12.715600 and the value of q i.e. – 12.7156 is printed as –12.715600.

- Second Group – The %10f indicates that width of 10 columns will used to print the floating point number. The number will be right justified in the given width by leaving required number of blank spaces at left. The value of p i.e. 12.7156 when printed will be 12.715600 and will occupy 9 columns and hence one blank space will be left out before the 12.715600.

  The value of q i.e. –12.7156 when printed will be –12.715600 which will occupy 10 columns and hence no blank spaces will be left out before the number.

- Third Group– When the floating point number is printed using %.2f notation, then two digits are printed after the decimal point. Hence, the value of p i.e. 12.7156 is rounded as 12.72 and printed

  The value of q i.e.–12.7156 is rounded as –12.72 and printed.

- Fourth Group– 10.2f indicates that total width of 10 columns will used to print the floating point number and two digits will be printed after the decimal point. The number will be right justified in the given width by leaving required number of blank spaces at left. The values of p i.e. 12.7156 will be rounded to 12.72 which will require 5 columns and hence 10–5=5 blank spaces are left out first and then the value is printed. The value of q i.e. –12.7156 when be rounded to –12.72 which will occupy 6 columns and hence 10–6=4 blank spaces will be left out before the number.

- Fifth Group – The %010.2f indicates that total width of 10 columns will used to print the floating point number and two digits will be printed after the decimal point. The number will be right justified in the given width and the required number of zeroes will be printed at left. The value of p i.e. 12.7156 will be rounded to 12.72 which will require 5 columns and hence 10 – 5 = 5 zeroes are printed first and then the values 12.72.

  The value of q i.e. – 12.7156 when be rounded to –12.72. The sign is printed first. The magnitude 12.72 will require 5 columns and hence 10 – 5 = 5 zeroes are printed first and then the value 12.72.

- Sixth Group – When the floating point number is printed using %.3f notation, then three digits are printed after the decimal point. Hence, the value of p i.e. 12.7156 is rounded as 12.716 and printed.

  The value of q i.e. – 12.7156 is rounded as –12.716 and printed

- Seventh Group – When the floating point number is printed using %05.3f notation, then three digits are printed after the decimal point. Hence, the value of p i.e. 12.7156 is rounded as 12.716 which will take 6 columns and hence the total width mentioned which is of 5 columns is less and hence ignored. Similar explanation is application for the value of q.

CWP6– Write the output of the following program.

Ans.– Refer you notebook for the program and the output.

Explanation–

- First Group – When the floating point number is printed using %e notation, then six digits are compulsorily printed after the decimal point. There will be exactly one non-zero digit before the decimal point. The decimal point will take one column. The exponent sign **e** will take one column. One more column will be taken by plus/minus sign of exponent part and the two columns will be taken by the exponent value. Hence, the value of p i.e. 12.7156 is printed as 1.27156e+01 and the value of q i.e. –12.7156 is printed as, –1.271560e+01.

- Second Group – When the floating number is printed using %2e notation, then two digits are printed after decimal point. The number will be rounded to two digits after the decimal point. Hence, the value of p i.e. 12.7156 is printed as 1.27e+01 and the value of q i.e. –12.7156 is printed as –1.27e+01.

- Third Group – When the floating point number is printed using %15e notation, then six digits are compulsorily printed after the decimal point. The decimal point will take one column. There will be exactly one non-zero digit before the decimal point. The exponent sign **e** will take one column. One more column will be taken by plus/minus sign of exponent part and the two columns will be taken by the exponent value.

  Hence, the value of p i.e. 12.1756 is printed as 1.271560e+01 will occupy 12 columns. Hence 15 – 12 = 3 blank spaces will be left out first and then the number is printed.

  The value of q –12.7156 is printed as –1.271560e+01 will occupy 13 columns.

  Hence 15 – 13 = 2 blank spaces will be left out first and then the number is printed.

- Fourth Group – When the floating point number is printed using %15.2e notation, then two digits are compulsorily printed after the decimal point. Hence, the value of p i.e. 12.7156 is printed as 1.27e+01 will occupy 8 columns. Hence 15 – 8 = 7 blank spaces will be left out first and then the number is printed.

  The value of q, i.e. –12.7156 is printed as –1.27e+01 will occupy 9 columns. Hence 15 – 9 = 6 blank spaces will be left out first and then the number is printed.

- Fifth Group – When the floating point number is printed using %015.2e notation, then two digits are compulsorily printed after the decimal point. Hence, the value of p i.e. 12.7156 is printed as 1.27e+01 will occupy 8 columns. Hence 15 – 8 = 7 zeroes are printed first and then the number is printed.

  The value of q i.e. –12.7156 is printed as –1.27e+01 will occupy 9 columns. The magnitude i.e. 1.27e+01 will occupy 8 columns. While printing, the sign is printed first then the 15 – 8 = 7 zeroes are printed and then the number 1.27e+01 is printed.

Dealing with the numbers is decimal, octal and hexadecimal numbers–

Using Comments in the program –

CWP7– Write the output of the following program.

Ans– Refer your notebook for the program and the output.

Explanation –

- The above program deals with decimal, octal and hexadecimal integers.
- In the above program, we are using comments. The comments are used for documentation purpose. The comments are not complied by the complier i.e. the comments are ignored during the compilation process.

- **Single Line Comment –**

General Syntax

```
/* comment line */
```

Example

```
/* C programming is appearing easy */
```

- **Multi-Line Comment-**

General Syntax

```
/* comment line 1
   comment line 2
   .....
   comment line n*/
```

Example

```
/* C programming
is appearing
easy */
```

CWP8 – Write the output of the following program.

Ans – Refer you notebook for the program and the output.

CWP9 – Write the output of the following program.

Ans – Refer you notebook for the program and the output.

Exercise-1: If x, y and z are integer variable and if x = 10, y =6 and z=3 then evaluate following.

(a) x– – + y– – + – –z/(x*x) (b) – – x + – –y* – –z

Ans: Refer you notebook for the program

Exercise-2 If a, b and c integer variable and if a = 2, b = 3 and c = 4 then evaluate following

(a) a = a++ + ++b +c; (b) b = ++a + b++ – ++c;

(c) x = ++a*b + (c++ ==4); (d) a = ++a + b++ + c;

Ans: Refer your notebook for the program.

CWP–10: Write the output of the following program.

Ans: Refer your notebook for the program.

CWP–11: Write the output of the following program.

Ans: Refer your notebook for the program

CWP–12: Write the output of the following program.

Ans: Refer your notebook for the program

CWP–13: Write the output of the following program.

Ans: Refer your notebook for the program

CWP–14: Write the output of the following program.

Ans: Refer your notebook for the program

CWP–15: Write the output of the following program.

Ans: Refer your notebook for the program

CWP–16: Write the output of the following program.

Ans: Refer your notebook for the program

5. EXPRESSING ALGORITHMS – SELECTION

5.1 Introduction –

- As we have seen earlier, **decision making** and **selecting** a particular path out of many available paths is a common requirement while implanting algorithms. Following are control structures/decision making / selection statements that are used in C.

  1) if statement
  2) if-else statement
  3) switch statement

5.2 if statement –

- The general syntax of if statement is given as follws.

```
if (conditional expression)
    statement
```

- If the conditional expression results in value true then the statement part will be executed. If the conditional expression results in value false then the statement part will not be executed.
- The statement can be any valid C statement
- Example–

```
if (m>=40)
    printf("Pass\n");
```

5.3 if–else statement –

- The general syntax of **if-else statement** is given as follows.

```
if (conditional expression)
    statement1
else
    statement2
```

- If the conditional expression results in value true then the statement1 will be executed and if the conditional expression results in value false then the statement2 will be executed.
- The statement can be any valid C statement.
- Example 1–

```
if (m>=40)
    printf("Pass\n");
else
    printf("Fail\n");
```

➤ Example 2–

```
if(m<40)
{
        printf("Fail\n");
        printf("Better luck next time\n");
}
else
{
        printf("Pass\n");
        printf("Congratulations\n");
}
```

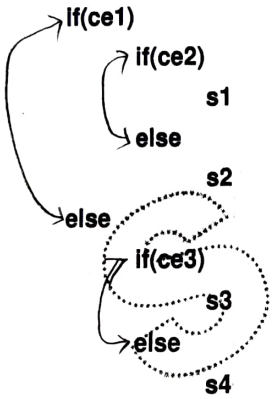
5.4 Nested if–else statement –

➤ When if statement or if-else statement itself acts as statement part of the other if or if-else statement then that result into nested if-else statement.

➤ There is no limit for the **depth of nesting.**

➤ The nested if-else statement can result into number of combinations. Consider the following example that shows one particular combination of the nested if-else statement.

➤ Example–



Explanation –

The following table explains the above nested if-else statement.

ce1	ce2	ce3	Executed Statement
false	false	false	s4
false	false	true	s3
false	true	false	s4
false	true	true	s3
true	false	false	s2
true	false	true	s2
true	true	false	s1
true	true	true	s1

The above table can also be represented using a **decision tree** as follow.

Note– When we have a complicate **Nested If Else** without any curly brackets, then to understand which else is associated with which if, we can apply the following logic.

Logic– Scan the Nested if else from top to bottom and search for the else. As soon as you get the else, associate it with the **previous**, **nearest** and **un-associated** if.

Solve the following examples, depending on the above logic.

Example 1:

```
if(ce1)
    if(ce2)
        s1
    else
        s2
```

ce1	ce2	Statement Executed
false	false	—
false	true	—
true	false	S2
true	true	S1

Example 2:

```
if(ce1)
    if(ce2)
        s1
else
    s2
```

ce1	ce2	Statement Executed
false	false	—
false	true	—
true	false	S2
true	true	S1

Example 3:

```
if(ce1)
{
    if(ce2)
        s1
}
else
    s2
```

ce1	ce2	Statement Executed
false	false	S2
false	true	S2
true	false	—
true	true	S1

Comment on above three examples –

In the above three example first and second are exactly same but in third due to curly braces there is only one statement i.e. If statement else is associated with If of ce1

CWP–17: Write a program that accepts one positive and prints whether the number is odd or even.

Ans: Try out on your own first and for additional reference use homework programs section.

CWP–18: Write a program that accepts one positive integer and prints whether the number is divisible by 7 or not.

Ans: Refer your notebook for the program.

CWP–19: Write a program that accepts one positive integer and prints whether the number ends with 7 or not.

Ans: Refer your notebook for the program.

CWP–20: Write a program that accepts two digit positive integer number and the program should calculate and print sum of the digits, product of the digits and the reverse number of the accepted number.

Ans: Refer your notebook for the program.

CWP–21: Write a program that accepts three digit positive integer number and the program should calculate and print sum of the digits, product of the digits and the reverse number of the accepted number.

Ans: Refer your notebook for the program.

CWP–22: Write a program that accepts a three digit positive integer and prints whether it is Armstrong number or not.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,x,y,z,w;
    clrscr();
    printf("Enter one positive integer=");
    scanf("%d", &n);
    if(n>=100&&n<=999)
    {
        x=n/100;
        w=n%100;
        y=w/10;
        z=w%10;
        if(x*x*x + y*y*y + z*z*z ==n)
            printf("%d is armstrong number\n",n);
        else
            printf("%d is not an armstrong number\n", n);
    }
    else
        printf("Invalid data\n");
    getch();
    return 0;
}
```

CWP–23: Write the output of the following program.

Ans: Refer your notebook for the program and the output.

CWP–24: Write a program that accepts marks of a single subject out of 100 and prints class of a student.

If marks>=70 Then the class is Distinction.

If marks>=60 AND if marks<70 Then the class is First Class.

If marks>=50 AND If marks<60 Then the class is Second Class.

If marks>=40 AND If marks<50 Then the class is Pass Class.

If marks < 40 Then the class is Fail.

Ans: Refer your notebook for the program.

CWP–25: Write a program that accepts three integers and finds out the highest out of them.

Ans: Refer notebook for the program.

CWP–26: Write a program that accepts one positive integer and the program should print a message that will display the divisibility and non-divisibility of number of 5 and 7.

Ans: Refer your notebook for the program.

CWP – 27: Write a program that will accept principal amount, period, age of the depositor, gender of the depositor and the membership status of the depositor. The program should calculate the final amount payable to the depositor. The rate of simple interest is decided as follows.

Gender	Age	Member	Rate of Interest
Male	< 60 yrs.	No	9.25%
		Yes	9.5%
	>= 60 yrs.	No	9.75%
		Yes	10.00%
Female	< 60 yrs.	No	9.35%
		Yes	9.6%
	>= 60 yrs.	No	9.85%
		Yes	10.10%

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     float p,r,amount;
     int n,age;
     char gender,member;

     clrscr();

     printf("Enter principle amount, age, member, period and gender=");
     scanf("%f %d %c %d %c",&p,&age,&member,&n,&gender);

     if(p<=0 || n<=0 || age<=0) printf("Invalid data\n");
     else
          if(gender!='m' && gender!='M' && gender!='f' && gender!='F')
               printf("Invalid gender\n");
          else
             if(member!='n' && member!='N' && member!='y' && member!='Y')
               printf("Invalid membership status\n");
          else
          {
               if(gender=='m'||gender=='M')
                    if(age<60)
                         if(member=='n'||member=='N')
                              r=9.25;
                         else
                              r=9.5;
                    else
                         if(member=='n'||member=='N')
                              r=9.75;
                         else
                              r=10.0;
               else
                    if(age<60)
                         if(member=='n'||member=='N')
                              r=9.35;
                         else
                              r=9.6;
                    else
                         if(member=='n'||member=='N')
                              r=9.85;
                         else
                              r=10.1;
               amount=p+p*n*r/100;
               printf("Final amount=%.2f\n",amount);
          }
     getch();
     return 0;
}
```

CWP-28: Write a program that accepts three numbers representing coefficients of a quadratic equation. Your program should print roots of the quadratic equation by considering all possibilities.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
     float a,b,c,d;
     clrscr();
     printf("Enter coefficients of quadratic equation=");
     scanf("%f %f %f",&a,&b,&c);
     if(a==0)
          if(b==0)
               printf("Invalid data\n");
          else
               printf("Root of linear equation=%f\n",-c/b);
     else
     {
          d=b*b-4*a*c;
          if(d==0) printf("Root=%f\n",-b/(2*a));
          else
          if(d>0)
          {
               printf("Root1=%f\n",(-b+sqrt(d))/(2*a));
               printf("Root2=%f\n",(-b-sqrt(d))/(2*a));
          }
          else
          {
               printf("Root1=%f + i(%f)\n",-b/(2*a), sqrt(-d)/(2*a));
               printf("Root2=%f - i(%f)\n",-b/(2*a), sqrt(-d)/(2*a));
          }
     }
     getch();
     return 0;
}
```

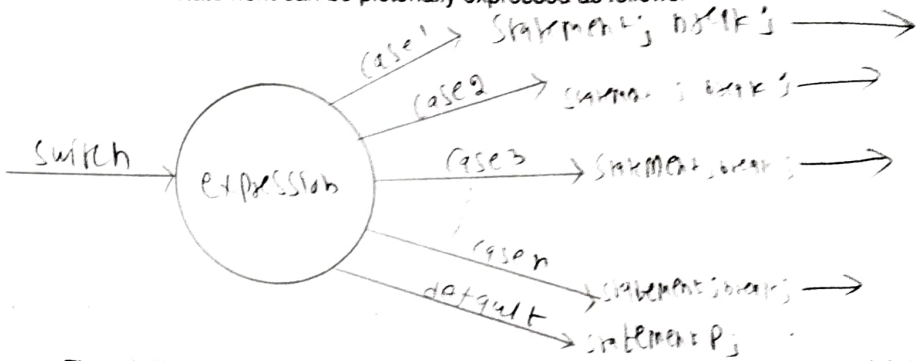
5.5 switch statement –

The general syntax of switch statement is as follows.

```
swich(expression)
{
     case 1 : statement 1; break;
     case 2 : statement 2; break;
     case 3 : statement 3; break;
     .....
     .....
     case n : statement n; break;
     default : statement p;
}
```

Explanation–

- The switch statement can be pictorially expressed as follows.



- First of all, the expression that is written with the switch will be solved to result into a single value.
- If the value of the expression matches with **case1** then the **statement1** will be executed and the corresponding **break** statement will take the control out the switch statement.
- If the value of the expression is not matching with **case1** then it will be compared with **case2** and if the match is found then the **statement2** will be executed and the corresponding **break** statement will take the control out of the switch statement.
- If the value of the expression is not matching with **case2** then it will be compared with further case and so on. If the value of the expression does not match with any of the cases then the **default** statement i.e. the **statement p** will be executed. Since the default part is physically written at the end of the switch statement, the control will come out of switch statement without writing the break statement with the default part.
- **Note1**– The default part of the switch statement need not be written at the end of the switch. It means we can write it at the beginning of the switch or it can be written anywhere in between within the switch.
- **Note2**– Writing the default part within the switch statement is not compulsory.

Consider the following examples.

Example1

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        case 1 : printf("10\n"); break;
        case 2 : printf("20\n"); break;
        case 3 : printf("30\n"); break;
        case 4 : printf("40\n"); break;.
        default : printf("100\n"); break;
}
```

Output–

x	1	2	3	4	Anything other than 1 to 4
Output	10	20	30	40	100

Example2–

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        case 1 : printf("10\n"); break;
        case 2 : printf("20\n");
        case 3 : printf("30\n"); break;
        case 4 : printf("40\n");
        default : printf("100\n");
}
```

Output–

x	1	2	3	4	Anything other than 1 to 4
Output	10	20 30	30	40 100	100

Example 3–

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        case 1 : printf("10\n");
        case 2 : printf("20\n");
        case 3 : printf("30\n"); break;
        case 4 : printf("40\n");
        default : printf("100\n");
}
```

Output–

x	1	2	3	4	Anything other than 1 to 4
Output	10 20 30	20 30	30	40 100.	100

Example 4 –

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        case 1 : printf("10\n");
        case 2 : printf("20\n");
        case 3 : printf("30\n");
        case 4 : printf("40\n");
        default : printf("100\n");
}
```

Output–

x	1	2	3	4	Anything other than 1 to 4
Output	10 20 30 40 100	20 30 40 100	30 40 100	40 100	100

Example 5 –

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        default : printf("100\n");
        case 1 : printf("10\n"); break;
        case 2 : printf("20\n");
        case 3 : printf("30\n"); break;
        case4 : printf("40\n");
}
```

Output–

x	1	2	3	4	Anything other than 1 to 4
Output	10	20 30	30	40	100

Example 6 –

```
int x;
printf("Enter one integer=");
scanf("%d", &x);
switch(x)
{
        case 1 : printf("10\n"); break;
        case 2 : printf("20\n");
        default : printf("100\n");
        case 3 : printf("30\n"); break;
        case4 : printf("40\n");
}
```

Output –

x	1	2	3	4	Anything other than 1 to 4
Output	10	20 30 100	30	40	100 30

CWP–29: Write a program that will ask user to enter one digit and depending on the digit entered by user the program should display the suitable message.

If the digit is 2 or 3 or 6 or 7 then the message should be **"Welcome".**

If the digit is 1 or 4 or 5 or 8 then the message should be **"Hello"**

If the digit is 0 or 9 then the message should be **"HI"**

If user enters anything other than 0 to 9 then the message should be **"Get Out"**

Ans: Refer your notebook for the program.

Explanation –

- As shown in the above program, t is possible to write the case numbers which are not specified in the order.
- Not writing the break statement in front of every case is used with advantage in the above program.

CWP–30: Write a program that accepts on Alphabet and prints the Colour name starting with that alphabet.

Ans: Refer your notebook for the program.

Explanation –

- Within the **switch** statement the case values need not be always integer numbers. It is possible to use **ASCII character constants** as **case values** in the above program.

CWP-31: Write a program that accepts option from user. The options can be '+' or '–' or '*' or '/'. After accepting the option, the program should accept the two integers and should print the appropriate result depending on the option.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int x,y;
     char option;
     clrscr();
     printf("Enter option out of + - * or / =");
     scanf("%c",&option);

     printf("Enter two integers=");
     scanf("%d %d",&x,&y);
     switch(option)
     {
          case '+':printf("%d\n",x+y);
                   break;
          case '-':printf("%d\n",x-y);
                   break;
          case '*':printf("%d\n",x*y);
                   break;
          case '/':
                   if(y!=0) printf("%f\n",(float)(x)/y);
                   else printf("Divide by zero error\n")
                   break;
          default   :printf("Invalid option\n");
                    break;
     }
     getch();
     return();
}
```

6.1 Introduction –

- The task which is being repeated is termed as **Iterative** task. **Repetition** of a particular task is a common process. For example, read the marks of 50 students and print the class of each student. In this situation the basic task is read marks of a single student and depending on the marks decide and print the class of a student. This basic task is to be repeated 50 times resulting into 50 **iterations.**

- A particular task might be repeated fixed number of times (unconditional iteration/looping) or the repetition might be dependent on a specific condition (conditional iteration / looping). Following are the control structures that are used in C for iteration.

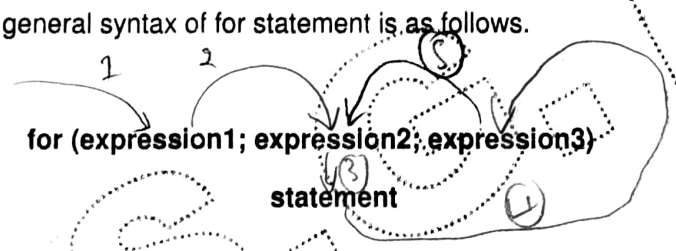
  (1) for statement

  (2) while statement

  (3) do-while statement

6.2 for statement –

- The general syntax of for statement is as follows.



for (expression1; expression2; expression3)

statement

- The expression1 is called as initialization expression. The expression2 is called conditional expression. The expression3 is called as modifier expression.
- The statement can be any valid C++ statement.
- Initially the control of execution will come to the expression1 and it will be executed. After that the control of execution will come to the expression2 and the expression2 is evaluated.
- If the expression2 is false in the very first comparison, then the control will come out of for loop without executing the statement part at all. If the expression2 results in value true then the statement part will be executed and after executing the statement part the control goes to the expression3 and it is executed.
- After executing the expression3, the control comes back to the expression2 and it will be executed. If the expression2 is true again, then the statement part will be executed again and the control will go back to expression3 again and it will be evaluated. The control will then be back to the expression2.
- The above process will be repeated as long as the expression2 is true and as soon as the expression2 becomes false then the control will come out of the for loop.

Example1 –
`for(i=1; i<=5; i++)`
`printf("%d\n", i);`
	Output – 1 2 3 4 5

Example2 –
`for(i=7; i<=5; i++)`
`printf("%d\n", i);`
	Output – NO OUTPUT

Example3 –
`for(i=1; i<=5; i=i)`
`printf("%d", i);`
	Output – 1 1 1 1 1 1 1 1 1 1 ...

Example4 –
`for(i=5; i>=1; i--)`
`printf("%d\n", i);`
	Output – 5 4 3 2 1

Example5 –
`for(i=1; i<=5; i++)`
`printf("%d\n", 23*i);`
	Output – 23 46 69 92 115

Example6 –
`char i;`
`for(i='A'; i<='E'; i++)`
`printf("%c\n", i);`
	Output – A B C D E

All the Possible Variations of writing the for loop –

(1)
`for(expression1; expression2; expression3)`
`statement`
	(2)
	`expression1;`
	`for(;expression2; expression3)`
	`statement`

(5)
`for (expression1; expression2;)`
`{`
`statement`
`expression3;`
`}`
	(4)
	`expression1;`
	`for(;expression2;)`
	`{`
	`statement`
	`expression3;`
	`}`

Infinite loop –

The for loop has one more special variation as follows.

always True condition

use for Infinite loop

```
for(;;)
{
.....
.....
}
```

In the above for loop, statement part is executed infinite number of times. This is because the expression2 i.e. the conditional expression is permanently true. The only way to come out of this loop is the use of break statement.

CWP–32: Write a program that accepts one positive integer say n and prints sum of the all the first n positive integers.

Ans: Refer your notebook for the program.

CWP–33: Write a program that accepts one positive integer say n and prints sum of all the odd number upto n.

Ans: Refer your notebook for the program.

CWP–34: Write a program that accepts one positive integer say n and prints sum of all the first n odd numbers.

Ans: Refer your notebook for the program.

CWP–35: Write a program that accepts one positive integer n and your program should calculate and print the answer of following series.

$$S = 1 - 2 + 3 - 4 + 5 - 6 + 7 - \ldots\ldots\ldots\ldots\text{upto n terms}$$

Ans: Refer your notebook for the program.

CWP–36: Write a program that accepts one positive integer n and your program should calculate and print the answer of following series.

$$S = \frac{1}{2} + \frac{3}{4} + \frac{5}{6} + \frac{7}{8} \ldots\ldots\ldots \text{upto n terms.}$$

Ans: Refer your notebook for the program.

CWP–37: Write a program that accepts one non negative integer and your program should print its factorial.

Ans: Refer your notebook for the program.

CWP–38: Write a program that accepts two positive integers and your program should print their gcd and lcm.

Ans: Refer your notebook for the program.

CWP–39: Write a program that accepts the integer value say n and then the program should accept n number of numbers. Our program should calculate and print how many of them are positive, how many of them are negative and how many of them are zeroes. The program should also calculate sum and the average of positive numbers independently and sum and average of negative number independently.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i,x,sp=0,sn=0,cp=0,cn=0,cz=0;
    clrscr();
    printf("How many integers=");
    scanf("%d",&n);
    if(n<=0) printf("Invalid data\n");
    else
    {
        printf("Enter data elements=");
        for(i=1;i<=n;i++)
        {
            scanf("%d",&x);
            if(x>0)
            {
                sp=sp+x;
                cp++;
            }
            else
            if(x<0)
            {
                sn=sn+x;
                cn++;
            }
            else cz++;
        }
        printf("Sum of positive elements=%d\n",sp);
        printf("Number of positive elements=%d\n",cp);
        if(cp!=0)
        printf("Avg. of positive elements=%f\n",float(sp)/cp);
        printf("Sum of negative elements=%d\n", sn);
        printf("Number of negative elements=%d\n", cn);
        if(cn!=0)
        printf("Avg. of negative elements=%f\n",float(sn)/cn);
            printf("Number of zeroes=%d\n", cz);
        }
        getch();
        return 0;
    }
```

CWP–40: Write a program that accepts one positive integer say n and program should calculate and print the answer of following series.

S = 2 + 4 + 8 + 16 + 32 + 64 +..... upto n terms

Ans:

Method–1:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
     int sum=0,i,n;
     clrscr();
     printf("How many terms=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
     {
          for(i=1;i<=n;i++)
               sum=sum+(int)(pow(2,i));
          printf("Answer=%d\n",sum);
     }
     getch();
     return 0;
}
```

Method–2:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int term=2,sum=2,i,n;
     clrscr();
     printf("How many terms=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
     {
          for(i=2;i<=n;i++)
          {
               term=term*2;
               sum=sum+term;
          }
          printf("Answer=%d\n",sum);
     }
     getch();
     return 0;
}
```

n = 5
i = 2

CWP–41: Write a program that accepts angle in degrees from user and will fine out the sine of that angle with the help of the following series.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \text{ ..... upto n terms. Where, x is radians}$$

Ans: Refer your notebook for the program.

CWP–42: Write a program that accepts a float value say x and one integer value say y which is negative or zero or positive. The program should calculate and print x^y without using library function.

Ans: Refer your notebook for the program.

CWP–43: Write a program that accepts principal amount, rate of interest and period in years. The amount is compounded annually. The program should calculate and print the amount that will be recoverable after given period.

$$A = P*(1+R/100)^N$$

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      float p,r,ans=1;
      int n,i;
      clrscr();
      printf("Enter principal amount, period and rate of interest=");
      scanf("%f %d %f",&p,&n,&r);
      if(p<=0||n<=0||r<=0) printf("Invalid data\n");
      else
      {
            for(i=1;i<=n;i++)
                  ans=ans*(1+r/100);
            printf("Final amount=%f\n",p*ans);
      }
      getch();
      return 0;
}
```

CWP–44: Write the output of the following program.

Ans: Refer your notebook for the program and the output.

CWP–45: Write the output of the following program.

Ans: Refer your notebook for the program and the output.

CWP–46: Write a program that finds four digit perfect squares, where the number represented by the first two digits and the last two digits are also perfect squares.

e.g. $1681 = 41^2$, $16 = 4^2$, $81 = 9^2$

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
     int n,x,y,i,j,k;
     clrscr();
     printf("Following are the four digit perfect square\n");
     for(n=1000;n<=9999;n++)
     {
          x=n/100;
          y=n%100;

          i=(int)(sqrt(n));
          j=(int)(sqrt(x));
          k=(int)(sqrt(y));

          if(i*i==n&&j*j==x&&k*k==y)
               printf("%d\n",n);
     }
     getch();
     return 0;
}
```

CWP–47: Write a program that accepts one positive integer n and program should print first n terms of the fibbonaccii series. Program should also print sum of all these terms. The series is given as,

0 1 1 2 3 5 8 13 21 34 55 89 .....

Ans: Refer your notebook for the program.

CWP–48: Write a program that prints all three digit Armstrong numbers.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int x,y,z,w,n;
     clrscr();
     printf("Following are all three digit armstrong numbers\n");
     for(n=100;n<=999;n++)
     {
          x=n/100;   //most significant digit
          w=n%100;
          y=w/10;    //middle digit
          z=w%10;    //least significant digit
          if(x*x*x+y*y*y+z*z*z==n)
               printf("%d\n",n);
     }
     getch();
     return 0;
}
```

CWP–49: Write a program to print the following output.

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

Ans: Refer your notebook for the program

CWP–50: Write a program to print the following output.

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

Ans: Refer your notebook for the program

CWP–51: Write a program to print the following output

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

Ans: Refer your notebook for the program

CWP–52: Write a program to print the following output.

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

Ans: Refer your notebook for the program

CWP–53: Write a program to print the following output.

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

Ans: Refer your notebook for the program

CWP–54: Write a program to print the following output.

	1							
	2		3					
	3		4		5			
	4		5		6		7	
	5		6		7		8	9

Ans: Refer your notebook for the program

CWP–55: Write a program to print the following output.

	1		0		1		0		1
	0		1		0		1	
	1		0		1			
	0		1					
	1							

Ans: Refer your notebook for the program

CWP–56: Write a program to print the following output.

	1								
	2		3						
	4		5		6				
	7		8		9		10		
	11		12		13		...		...

Upto n, where n is a positive integer entered by user.

Ans: Refer your notebook for the program

CWP–57: Write a program to print the following output.

									1
							2		3
					3		4		5
			4		5		6		7
	5		6		7		8		9

Ans: Refer your notebook for the program

CWP–58: Write a program to print the following output.

									1									
							2		3		2							
					3		4		5		4		3					
			4		5		6		7		6		5		4			
	5		6		7		8		9		8		7		6		5	

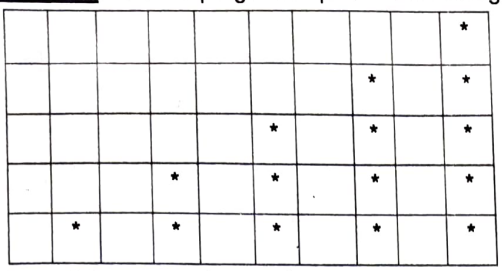
Ans: Refer your notebook for the program

CWP–59: Write a program to print the following output.

									1									
							2		3		2							
						3		4		5		4		3				
					4		5		6		7		6		5		4	
			5		6		7		8		9		8		7		6	
	6		7		8		9		0		1		0		9		8		7		6

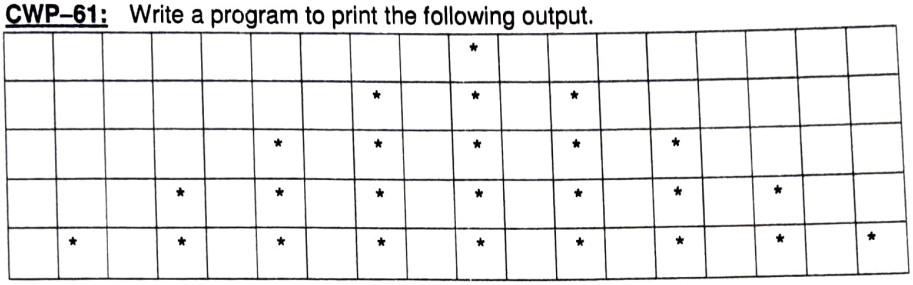
Ans: Refer your notebook for the program

CWP–60: Write a program to print the following output.



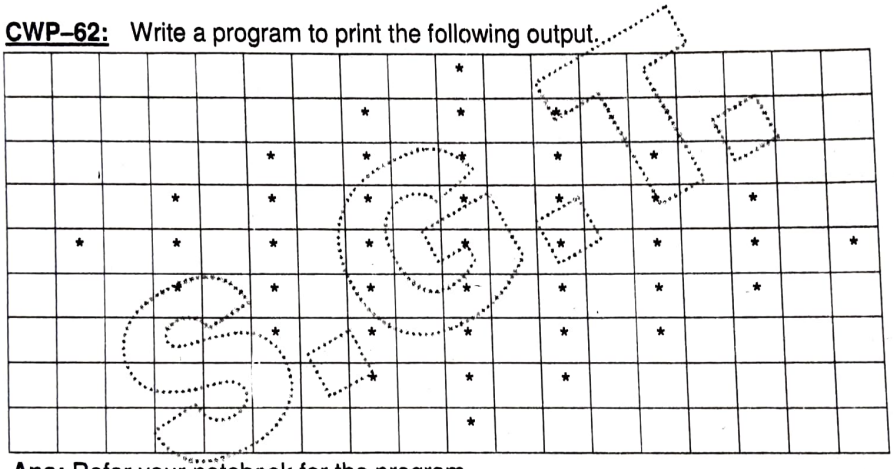
Ans: Refer your notebook for the program

CWP–61: Write a program to print the following output.



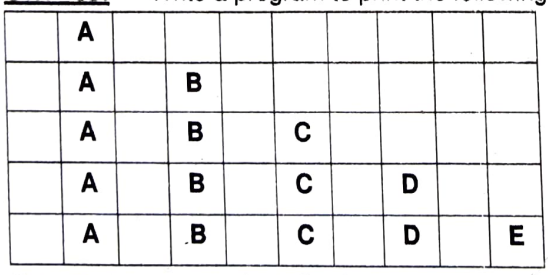
Ans: Refer your notebook for the program

CWP–62: Write a program to print the following output.



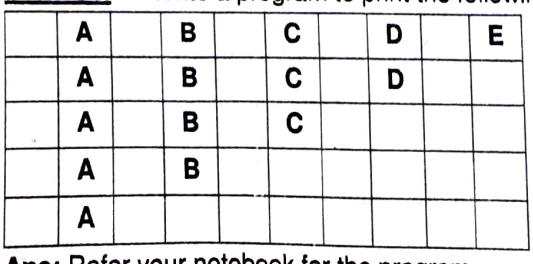
Ans: Refer your notebook for the program

CWP–63: Write a program to print the following output.



Ans: Refer your notebook for the program

CWP–64: Write a program to print the following output.



Ans: Refer your notebook for the program

CWP–65: Write a program to print the following output.

	E								
	E		D						
	E		D		C				
	E		D		C		B		
	E		D		C		B		A

Ans: Refer your notebook for the program

CWP–66: Write a program to print the following output.

	E		D		C		B		A
	E		D		C		B		
	E		D		C				
	E		D						
	E								

Ans: Refer your notebook for the program.

CWP–67: Write a program to print the following output.

	E		F		G		H		I
	F		G		H		I		
	G		H		I				
	H		I						
	I								

Ans: Try out on your own first and for additional reference use homework programs section.

CWP–68: Write a program to print the following output.

	*		*		*		*		*		*		*		*	
	*		*		*		*				*		*		*	
	*		*		*								*		*	
	*		*												*	
	*															*

Ans: Refer your notebook for the program.

CWP–69: Write a program to print the following output.

	A		B		C		D		E		D		C		B		A
	A		B		C		D				D		C		B		A
	A		B		C								C		B		A
	A		B												B		A
	A																A

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     char i,j,m;
     int k;
     clrscr();
     for(i='E';i>='A';i- -)
     {
          for(j='A';j<=i;j++)
               printf("%c",j);

          for(k=1;k<=4*(69-(int)(i))-2;k++)
               printf(" ");

          if(i=='E') m='D';
          else m=i;

          for(j=m;j>='A';j- -)
               printf("%c",j);

          printf("\n");
     }
     getch();
     return 0;
}
```

CWP–70: Write a program to print the following output.

					1				
				1		1			
			1		2		1		
		1		3		3		1	
	1		4		6		4		1

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int n,r,i,k;
     int nf,rf,nrf;
     clrscr();
     for(n=0;n<=4;n++)
     {

          for(k=1;k<=4-n;k++)
               printf(" ");

          nf=1;
          for(i=1;i<=n;i++)
               nf=nf*i;

          for(r=0;r<=n;r++)
          {
               rf=1;
               for(i=1;i<=r;i++)
                    rf=rf*i;


               nrf=1;
               for(i=1;i<=n-r;i++)
                    nrf=nrf*i;

                         214
               printf("%2d",nf/(rf*nrf));
          }
          Printf("\n");
     }
     getch();
     return 0;
}
```

CWP–71: Write a program to print the following output.

					1				
				1	2	A			
			1	2	3	A	B		
		1	2	3	4	A	B	C	
	1	2	3	4	5	A	B	C	D

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int i,j,k;
      char m;
      clrscr();

      for(i=1;i<=5;i++)
      {
            for(k=1;k<=5-i;k++)
                  printf(" ");

            for(j=1;j<=i;j++)
                  printf("%d",j);

            for(m='A';m<='A'+i-2;m++)
                  printf("%c",m);

            printf("\n");
      }
      getch();
      return 0;
}
```

6.3 while statement–

- The general syntax of the while loop or while statement is as follows.

  while(conditional expression)

  statement

- First of all, the control of execution will come to the conditional expression and it will be executed. If the conditional expression is false in the very first comparison then the control of execution will come out of the while loop without executing the statement part at all.
- If the conditional expression results in value true then the statement part will be executed. After executing the statement part, the control of execution will be back to the conditional expression and it will be re-evaluated. If it is true, again the statement part will be executed the control of execution will be back to the conditional expression again.
- This process will be repeated as long as the conditional expression is true. As soon as, the conditional expression becomes false, the control of execution will come out of the while loop.

Example1 –

```
i =1;
while(i<=5)
{
   printf("%d\n", i);
    i++
}
```

Output –

1
2
3
4
5

Example2 –

```
i=1;
while(i<=5)
   printf("%d\n", i++);
```

Output –

1
2
3
4
5

Example3 –

```
i=1;
while(i<=5)
   printf("%d\n", ++i);
```

Output –

2
3
4
5
6

Example4 –

```
i=7;
while(i<=5)
   printf("%d\n", ++i);
```

Output –

No output

Example5 –

```
i=1;
while(i<=5)
   printf("%d\n", i);
```

Output –

1
1
1
1
:

Example6 –

```
i=5;
while(i>=1)
{
    printf("%d\n", i);
    i– –;
}
```

Output –

5
4
3
2
1

Example7 –

```
i=5;
while(i>=1)
 printf("%d\n", i– –);
```

Output –

5
4
3
2
1

Example8 –

```
i=5;
while(i>=1)
   printf("%d\n", – –i);
```

Output –

4
3
2
1
0

Conversion From for loop to while loop–

Example1:

Original loop:

```
for(i=1;i<=5;i++)
        printf("%d\n", i);
```

Converted loop:

```
i=1;
while(i<=5)
{
        printf("%d\n", i);
        i++;
}
```

Conversion process:

(a) First of all, write expression1 of the loop independently.

(b) Then write the while loop with the conditional expression as expression2 of the for loop.

(c) Append the expression3 of for loop to the statement part and enclose the statement part and expression3 within the pair of curly brackets and associate that pair with the while loop.

Convert following for loop to equivalent while loops.

Exercise–1:

```
fact=1;
for(i=1;i<=n;i++)
        fact=fact*i;
printf("Factorial of %d=%ld\n", n, fact);
```

```
fact = 1;
i = 1;
while (i<=n)
{fact = fact * i
i++
}
printf ("Factorial of %d = %ld", n, fact);
```

Exercise–2:

```
sum=f+s;
printf("%d %d", f, s);
for(i=3;i<=n;i++)
{
        t=f+s;
        printf("%d ",t);
        sum=sum+t;
        f=s;
        s=t;
}
print("\nSum=%d\n", sum);
```

```
s = f + s;
printf("%d %d", f, s);
i = 3;
while (i<=n)
{ t = f + s;
  printf("%d", t);
  sum = sum + t;
  f = s;
  s = t
  i++
}
printf ("\n Sum = %d", sum);
```

Note: All the programs that we have written with the for loop can be easily written with the while loop with the help of conversion logic.

Conversion From Nested for loop to Nested while loop –

Example1:

Original Nested loop:

```
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
        printf("%2d", j);
    printf("\n");
}
```

Converted Nested loop:

```
i=1;
while(i<=5)
{
    j=1;
    while(j<=i)
    {
        printf("%2d", j);
        j++;
    }
    printf("\n");
    i++;
}
```

Conversion process:

(a) First of all, write the expression1 of the for loop independently.

(b) Then write the while loop with the conditional expression as expression2 of the for loop.

(c) Append the expression3 of for loop to the statement part and enclose the statement part and expression3 within the pair of curly brackets.

(d) Carry out the above process for inner **for** loop.

Convert following Nested for loops to equivalent Nested while loops.

Example–1

```
for(i=5;i>=1;i– –)
{
    for(j=1;j<=i;j++)
        printf("%2d", j);
    printf("\n");
}
```

Example–2

```
for(i=1;i<=6;i++)
{
    for(k=1;k<=2*(6–i);k++)
        printf(" ");
    for(j=i;j<=2*i–1;j++)
        printf("%2d", j%10);
    for(m=2*i–2;m>=i;m– –)
        printf("%2d", m%10);
    printf("\n");
}
```

Note: All the programs that we have written with the nested for loops can be easily written with the nested while loops with the help of conversion logic.

CWP–72: Write a program that accepts one positive integer and the program should print the sum of the digits, product of the digits present in the number and the reverse of the given number.

Ans: Refer the notebook for the program.

CWP–73: Write a program that accepts one positive integer say n and the program should print whether the number n is prime or not.

Ans: Refer the notebook for the program.

CWP–74: Write a program that accepts one positive integer and will print the number in such a way that every digit will be replaced by the corresponding word.

For e.g. if the number→7482
then the output → seven four eight two.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int n;
     int df=1,y,digit;
     clrscr();
     printf("Enter one positive integer=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
     {
          y=n;
          while(n>9)
          {
               n=n/10;
               df=df*10;
          }
          while(df!=0)
          {
               digit=y/df;
               switch(digit)
               {
                    case 0:printf("zero "); break;
                    case 1:printf("one "); break;
                    case 2:printf("two "); break;
                    case 3:printf("three "); break;
                    case 4:printf("four "); break;
                    case 5:printf("five "); break;
                    case 6:printf("six "); break;
                    case 7:printf("seven "); break;
                    case 8:printf("eight "); break;
                    case 9 : printf("nine "); break;
               }
               y=y%df;
               df=df/10;
          }
     }
     getch();
     return 0;
}
```

CWP–75: A famous conjecture holds that all positive integers converges to 1(one) when treated as follows.

(1) If the number is odd, it is multiplied by three and one is added.

(2) If the number is even, it is divided by two.

(3) Continuously apply above operations to the intermediate results until the number converges to one.

Write a program to read one integer number from keyboard the implement the above mentioned algorithm and display all the intermediate value until the number converges to one. Also count and display number of iterations required for the convergence.

Ans: Refer the notebook for the program.

CWP–76: Write a program that accepts one positive integer and prints its prime factors.

Ans: Refer the notebook for the program

6.4 do–while statement–

- The general syntax of the do–while statement is as follows.

```
do
    statement
while(conditional expression);
```

- First of all, control of execution will come to the statement part and it will be executed. After executing the statement part, the control of execution will come to the conditional expression and it will be checked.

- If the conditional expression is false in the very first comparison then the control of execution will come out of the do–while loop.

- If the conditional expression is true, then the control of execution will go back to the statement part and it will be executed again. After executing the statement part, again the conditional expression is checked. If the conditional expression is true again then the control will go back to the statement part.

- The process will be repeated as long as the conditional expression is true. As soon as the conditional expression is false, the control of execution will come out of the do–while loop.

- **Note:** Since, the statement part is executed first and then the condition is checked , the statement part is always executed at least once.

Example 1 –

```
i=1
do
{
   printf("%d\n", i);
   i++;
}
 while(i<=5);
```

Output –

```
1
2
3
4
5
```

Example 2 –

```
I=1;
do
     printf("%d\n", I++);
while(I<=5);
```

Output –

```
1
2
3
4
5
```

Example 3 –

```
I=1;
do
   printf("%d\n", ++I);
while(I<=5);
```

Output –

```
2
3
4
5
6
```

Example 4 –

```
I=7;
do
     printf("%d\n", ++i);
while(i<=5);
```

Output –

```
8
```

Example 5 –

```
i=1;
do
   printf("%d", i);
while(i<=5);
```

Output –

```
11111-----
```

Example 6 –

```
I=5;
do
{
     printf("%d\n", i);
     i- -;
}
 while(i>=1);
```

Output –

```
5
4
3
2
1
```

Example 7 –

```
I=5;
do
     printf("%d\n", I- -);
while(I>=1);
```

Output –

```
5
4
3
2
1
```

Example 8 –

```
i=5;
do
     printf("%d\n", - -i);
while(i>=1);
```

Output –

```
4
3
2
1
0
```

Conversion from for loop to do-while loop –

Example 1–

Original loop:

```
for(i=1;i<=5;i++)
    printf("%d\n", i);
```

Converted loop:

```
i=1;
if(i<=5)
    do
    {
        printf("%d\n",i);
        i++;
    }
    while(i<=5);
```

Conversion process:

(a) First of all, write the expression1 of the loop independently.

(b) Since, the for loop then checks the condition, we have to write the if statement with conditional expression as expression2 of the for loop. This is because, in do-while, the condition is checked later loop. This is because, do-while, the condition is checked later whereas we want exactly equivalent logic.

(c) Then write the do-while loop with the conditional expression as expression2 of for loop. The do-while should act statement part of if statement.

(d) Append the expression3 of for loop to the statement part and enclose the statement part and expression3 within the pair of curly brackets and associate that pair with the do-while loop.

Convert following for loops to equivalent do-while loops.

Exercise–1:

```
fact=1;
for(i=1;i<=n;i++)
    fact=fact*i;
printf("Factorial of %d=%ld/n",n,fact);
```

Exercise–2:

```
sum = f + s;
printf("%d %d ",f,s);
for(i=3,i<=n;i++)
{
    t = f + s;
    printf("%d ", t);
    sum = sum + t;
    f = s;
    s = t;
}
printf("\nSum=%d\n", sum);
```

Note: All the programs that we have written with the for loops can be easily written with the do-while loops with the help of conversion logic.

Conversion from Nested for loops to Nested do-while loops–

Example1–

Original Nested loop:

```
for(i=1;i<=5;i++)
{
    for(j=1;j<=i;j++)
        printf("%2d", j);
    printf("\n");
}
```

Converted Nested loop:

```
i=1;
if(i<=5)
    do
    {
        j=1;
        if(j<=i)
            do
            {
                printf("%2d", j);
                j++;
            }
            while(j<=i)
        printf("\n");
        i++;
    }
    while(i<=5);
```

Conversion process:

(a) First of all, write the expression1 of the for loop independently.

(b) Since, the **for** loop then checks the condition, we have to write the if statement with conditional expression as expression2 of the **for** loop. This is because, in do-while loop, the condition is checked later whereas we want exactly equivalent logic.

(c) Then write the do-while loop with the conditional expression as expression2 of for loop. The do-while should act as statement part of if statement.

(d) Append the expression3 of for loop to the statement part and enclose the statement part and expression3 within the pair of curly brackets and associate that pair with the do-while loop.

(e) Carry out the above process for inner **for** loop.

Convert following Nested for loops to equivalent Nested do-while loops.

Exercise–1:

```
for(i=5;i>=1;i--)
{
    for(j=1;j<=i;j++)
        printf("%2d", j);
    printf("\n");
}
```

Exercise–2:

```
for(i=1;i<=6;i++)
{
    for(k=1;k<=2*(6–i);k++)
        printf(" ");
    for(j=i;j<=2*i–1;j++)
        printf("%2d", j%10);
    for(m=2*i–2;m>=i;m– –)
        printf("%2d", m%10);
    printf("\n");
}
```

Note: All the programs that we have written with the nested for loops can be easily written with the nested do-while loops with the help of conversion logic.

CWP-77: Write a menu driven program that will display the menu for the user and the menu is as follows.

1. **Addition**
2. **Subtraction**
3. **Multiplication**
4. **Division**
5. **Quit**

The program should read the option from user for the repeated execution until user enters option for quit.

Ans: Refer your notebook for the program.

CWP-78: Write a menu driven program for similar requirements as that of the previous with the only difference that the menu is changed as follows.

+	**Addition**
–	**Subtraction**
*	**Multiplication**
/	**Division**
Q,q	**Quit**

The program should read the option from user for the repeated execution until user enters option for quit.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int x,y;
     char option;
     do
     {
        clrscr();
        printf("+:Add\n");
        printf("-:Subtract\n");
        printf("*:Multiply\n");
        printf("/:Divide\n");
        printf("Q or q:Quit\n");
        printf("Enter your option=");
        scanf("%c",&option);
        if(option=='+'||option=='-'||option=='*'||option=='/')
        {
         printf("Enter two integers=");
         scanf("%d %d",&x,&y);
         switch(option)
         {
            case '+':printf("%d\n",x+y);break;
            case '-':printf("%d\n",x-y);break;
            case '*':printf("%d\n",x*y);break;
            case '/':if(y!=0) printf("%f\n",(float)(x)/y);
                     else printf("Divide by zero error\n");
                     break;
         }
         getch();
        }
     }
        while(option!='Q' && option!='q');
        return 0;
}
```

CWP-79: Write a program that accepts stream of integers. The stream ends with zero. The program should calculate and print how many of them are positive and negative. The program should also calculate and print sum of positive and sum of negative elements independently. The program should also calculate and print average of positive elements and average of negative elements independently.

Ans: Refer your notebook for the program.

Differentiate between while statement and do-while statement –

While statement	do while statement
1. The general syntax of the while statement is given as follows. **while(conditional expression)** **statement**	1. The general syntax of the do while statement is given as follows. **do** **statement** **while (conditional expression);**
2. The conditional expression is checked first and then the statement part is executed.	2. The statement part is executed first and then the conditional expression is checked.
3. In while, it is possible that the statement part is not at all executed.	3. In do while, the statement part is always executed at least once.
4. The while statement has entry as well as exit control.	4. The do-while no entry control. It has only exit control.
5. Example – **i = 7;** **while(i<=5)** **printf("%d",i);** The above example will not result into any output.	5. Example – **i = 7;** **do** **{** **printf("%d",i);** **}** **while(i<=5);** The above example will result into output as **7**
6. The while statement is made up of a keyword i.e. **while.**	6. The do-while statement is made up of two keywords named as **do** and **while.**
7. NO semicolon in front of while.	7. Semicolon is compulsory with do-while.

6.5 break statement –

- The **break** statement is used to take the control of execution **abruptly** out of the existing **control structure**.
  Thus, we can say that the break statement breaks the normal sequential flow.

- The break statement can be used within,

  a) switch statement
  b) for, while and do-while statement

break statement within switch statement–

Refer the topic, **switch statement** for examples and programs.

break statement within the loops –

When the break statement within the loop gets executed then the control of execution will come out of the loop abruptly. Consider the following examples.

Example1–

```
for(i=1;i<=100;i++)
        if(i>5)break;
        else printf("%d",i);
```

Output –

```
12345
```

Explanation –

When the value of i will become 6 then the break statement will be executed and the control of execution will come out of the for loop abruptly. We can avoid the break statement in the above example and can still get the same output. Thus, the same output can be obtained with the better logic, as follows.

Better Logic –

```
for(i=1;i<=5;i++);
        printf("%d", i);
```

Example2–

```
for(i=1;i<=5;i++)
{
        for(j=1;j<=100;j++)
                if(j>i)break;
                else printf("%2d", j);
        printf("\n");
}
```

Output–

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

Explanation –

In the above example, the break statement is used within the inner for loop. When the break statement in the above example gets executed, the control of execution will come out of the inner for loop and it will not come out of the outer for loop. The above example will result into the output as shown.

We can avoid the break statement in the above example and can still get the same output. Thus, the same output can be obtained with the better logic, as follows.

Better Logic –

```
for(i=1;i<=5;i++)
{
        for(j=1;j<=i;j++)
                printf("%2d",j);
        printf("\n");
}
```

Example3 –

```
scanf("%d",&n);
j=2;
while(j<n)
        if(n % j==0) break;
        else j++;
if(j < n)
        printf("%d is not prime\n",n);
else
        printf("%d is prime\n",n);
```

Explanation –

In the above logic, when n will be divisible by j then the break statement will be executed and the control of execution will come out of the while loop abruptly. It indicates that the number is not prime.

If n is not divisible by any of the values of j upto the n then the control of execution will come out of the while loop normally indicating that n is prime.

6.6 continue statement –

- The continue statement is used within the loops. When the continue statement within the loop gets executed, then the control of execution will **continue** to the next iteration of the loop. All the further statements in the current iteration of the loop are skipped.

- When the **continue** statement within the **for loop** gets executed then the control of execution continues to the **expression3** of the for loop.

- When the **continue** statement within the **while loop** gets executed then the control of execution continues to the **conditional expression** of the while loop.

- When the **continue** statement within the **do-while loop** gets executed then the control of execution continues to the **conditional expression** of the do-while loop.

- Consider the following example.

Example1 –

```
for (i=1;i<=100;i++)
{
        if (i%7!=0) continue;
        printf("%d ",i);
}
```

Output –

7 14 21 28 35 42 49 56 63 70 77 84 91 98

Explanation –

When the value of i is 1, then the condition **i%7!=0** is **true** and hence, the continue statement gets executed. Due to the execution of the continue statement, the control of execution will continue to the next iteration i.e. the expression3 of the for loop i.e. i++ and thus the printf( ) is skipped. Similar will be the action for all the values of i from 2 to 6.

When the value of i will become 7, then the condition **i%7!=0** will become **false** and hence the continue statement is not executed and the printf( ) statement is executed and the value 7 is printed and the control of execution will go to the next iteration normally.

The process will be repeated for the further iterations. During the process, all the multiples of 7 upto 100 will be printed. The same output can be obtained without using continue statement. The better logic to give the same output without using continue statement can be given as follows.

Better Logic –

```
for (i=1;i<=100;i++)
if (i% 7= =0) printf("%d ",i);
```

All the values for which the condition i%7==0 is true are printed i.e. all multiples of upto 100 are printed.

6.7 goto statement –

- The **goto** statement is used to take the control of execution abruptly anywhere in the program.
- The general syntax of the goto statement is as follows.

goto label

When the goto statement is executed, then the control of execution will go to the labelled statement.

Consider the following example.

```
fact = 1;
i = 1;
abc: fact = fact * i;
i++;
If (i<=n) goto abc;
printf("Factorial of %d = %lu\n", n, fact);
```

In the above example, if the condition **i<=n** is **true** then the statement **goto abc;** will be executed and hence the control of execution will jump to the statement with the **label abc** i.e. the statement **fact = fact * i;**.

Thus, in the above example, the **goto** statement causes back jump and thus implements the **loop action**.

When the condition, **i<=n** will be **false**, then the statement **goto abc** will not be executed and the control will come to the **printf( )** statement and it will be executed.

Note1 – The labelled statement and the label are separated with a **colon(:)**.

Note2 – Using goto statement, the **forward jump** as well as **back jump** can both be implemented.

Note3 – Since, the goto statement breaks the sequential flow of the program, using too many goto statements in the program is not advisable.

CAP-80: Write a program that accepts integer and print its factorial using goto statement. (without using loop)

Ans: Refer your notebook for the program.

6.8 Reading and Writing Character data –

- The reading and writing of character data is possible using the library functions getchar() and putchar(). To use these library functions, we have to include a file called as **stdio.h**.
- The function **getchar( )** is used to get a character i.e. to read a character from a file called as **stdin (standard input device file)**. The getchar( ) will read a character from stdin and it will return that character.
- The function **putchar( )** is used to put a character i.e. to write a character into a file called as **stdout (standard output device file)**.

CWP-81: Write a program that will read a character from user and will print the message indicating whether that character is an alphabet or digit or any other character.

Ans: Refer your notebook for the program.

CWP-82: Write a program that will read a character from user. The program should be able to give the message whether the character entered is vowel or consonant or digit or any other special character.

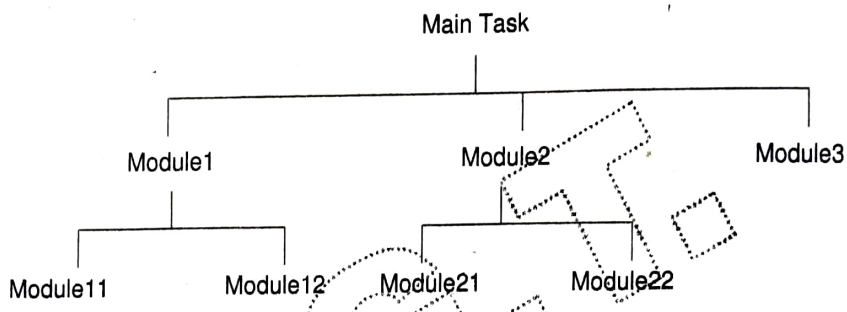
Ans: Refer your notebook for the program.

CWP-83: Write a program that will read a character from user. If the character is a uppercase letter then the program should print the corresponding lowercase letter and vice-versa and if it is any other character then it should be printed as it is.

7. DECOMPOSITION OF SOLUTION

7.1 TOP DOWN MODULAR PROGRAMMING –

- In case of small programming problems, the solution is also often small and straightforward. However, when the programming task is big in size, it is often divided logically into subtasks. These subtasks can also be divided further if required and so on. In other words the main task is divided into modules and the modules are divided into sub-modules and so on.
- Every sub-task (module) will have its own problem definition and it will have its own solution. Solutions of all the modules are implemented independent of each other and will then be joined together to result into overall solution.
- Consider the following diagram.



- As shown in the above diagram, the main task is at the top level and is divided into three modules named as Module1, Module2 and Module3. The Module1 is further divided into two modules named as Module11 and Module12. The Module2 is further divided into two modules named as Module21 and Module22. The division of modules and thus the planning of the overall project i.e. the main task is done from top to bottom and hence the above approach is called as **Top Down Modular Programming**.

- **Advantage of Top Down Modular Programming –**

1. The logical clarity of the problem increases right at the beginning of the development of the work. Hence the chances of making mistakes reduce.
2. Since the different modules can be developed in parallel by many programmers the project can be finished within the less period.
3. The debugging becomes easier. This is because it becomes easier to concentrate on the specific module by looking to the nature of logical errors.
4. The reusability is another significant advantage. It means that certain modules of the project can be used as it is or with little modifications for similar projects.

7.2 Fundamentals of Functions –

To understand these fundamentals, let us consider the following example.

```
#include>stdio.h>
void main()
{
    int x,y,z;
    int add(int, int);
    printf("Enter two integers=");
    scanf("%d %d",&x,&y);
    z = add(x, y);
    printf("Answer=%d\n",z);
}
int add(int p, int q)
{
    return p + q ;
}
```

- **Function Declaration or Function Prototype –**

Consider the following statement.

```
int add (int, int);
```

Using the above line we **declare** to the complier that add( ) is a function that accepts two integers and returns one integer.

- **Function Call –**

Consider the following,

```
z = add(x, y);
```

Using the statement, we are **calling the function add( )** from the main function. While calling the function the values of x and y are passed and are copied into p and q respectively. The function will add these values and will return the answer of addition. The returned answer will be assigned to z and will then the printed using printf( ).

- **Function Definition (header and body) –**

Consider the following example.

```
int add(int p, int q)
{
    return p + q;
}
```

The first line in the above example is called as **Function Header**. The general syntax of the function header is given das,

return_type function name (optional list of parameters)

The code of the function enclosed within the pair of curly brackets is called as **Function Body**. The function header and the function body together is called as **Function Definition**.

➢ **Actual Parameters (also called as Arguments) –**

The parameters which are used in the function call are called as actual parameters (arguments). For e.g. **x** and **y** are actual parameters (arguments).

➢ **Formal Parameters (also called as Parameters) –**

The parameters which are used in the function header are called as formal parameters (parameters). For e.g. **p** and **q** are formal parameters (parameters).

➢ **Points to note –**

1. When the function is called, the values of actual parameters are copied into the corresponding formal parameters.
2. The number of actual parameters and formal parameters must be same.
3. The actual parameters and the corresponding formal parameters must be same in type.
4. The actual parameters and formal parameters may have same name.
5. The actual parameters and formal parameters will occupy **separate** memory locations irrespective of whether they have same names or different names.
6. In the above points, we can use the terms Arguments and Parameters instead of Actual Parameters and Formal Parameters, respectively.

CWP-84: Write a function that accepts one integer and calculates and returns its factorial. Write a suitable main function to accept the values of n and r from user and will calculate and print the value of ${}^{n}C_{r}$ using above function.

Ans: Refer your notebook for the program.

CWP-85: Write a function that accept three integers and returns the highest out of them. Write a program that will accept 9 integers from user and will print the highest out of them using the given function.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int a,b,c,d,e,f,g,h,i,m1,m2,m3,m;
      int highest(int,int,int);
      clrscr();
      printf("Enter 9 integers=");
      scanf("%d %d %d %d %d %d %d %d %d",&a,&b,&c,&d,&e,&f,&g,&h,&i);
      m1=highest(a,b,c);
      m2=highest(d,e,f);
      m3=highest(g,h,i);
      m=highest(m1,m2,m3);
      printf("Highest=%d\n",m);
      getch();
      return 0;
}
int highest(int x,int y,int z)
{
      return x>y?x>z?x:z:y>z?y:z;
}
```

CWP-86: Write a function that accepts three digit number. The function should determine and return the answer indicating whether than number is Armstrong number or not. Write a suitable main function that will print all the three digit Armstrong numbers using the above function.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int n,ans;
      int armstrong(int);
      clrscr();
      printf("Following are all three digit armstrong numbers\n");
      for(n=100;n<=999;n++)
      {
            ans=armstrong(n);
            if(ans==1) printf("%d\n",n);
      }
      getch();
      return 0;
}
int armstrong(int n)
{
      int x,y,z,w;
      x=n/100;
      w=n%100;
      y=w/10
      z=w%10;
      if(x*x*x+y*y*y+z*z*z==n) return 1;
      else return 0;
}
```

CWP-87: Write a function that accepts integer number. The function should determine and return the answer indicating whether that number is prime number or not. Write a suitable main function that will print all the prime numbers between 1 to n using the above function, where n is accepted from user.

Ans: Refer your notebook for the program.

CWP-88: Write the output of the following program.

Ans: Refer your notebook for the program.

7.3 Storage Classes –

The variable declared in the program will have certain storage class. The storage classes that are used in C++ are namely **auto**, **static**, **register** and **extern**.

7.3.1 auto and static storage class –

Consider the following pair of programs.

Program-1

```
#include<stdio.h>
int main()
{
    int i;
    void fun();
    for(i=1;i<=5;i++)
        fun();
    return 0;
}
void fun()
{
    auto int x=1;
    printf("%d\t",x);
    x++;
}
```

Program-2

```
#include<stdio.h>
int main()
{
    int i;
    void fun();
    for(i=1;i<=5;i++)
        fun();
    return 0;
}
void fun()
{
    static int x=1;
    printf("%d\t",x);
    x++;
}
```


➢ **Explanation for Program-1:**

In program-1, the storage class of 'x' is **auto**. When the function fun( ) is called for the first time then the compiler will allocate space of 2 bytes for 'x' and will initialize it with the value 1. The allocation will take place **automatically.**

The value of 'x' i.e. 1 will be printed and then it will be incremented to 2. The function call will then be finished. The complier will then **de-allocate** the 'x' along with its value **automatically** and the control will return back to the main( ).

The function fun( ) will be called again. The compiler will then re-allocate the 'x' **automatically** and will re-initialize its value with 1. The value of 'x' i.e. 1 will be printed and then 'x' will become 2. The function will be finished and then the 'x' will be de-allocated **automatically** along with its value. The process will be repeated and the output will be,

1 1 1 1 1

➢ **Explanation for Program-2:**

In program-2, the storage class of 'x' is **static**. When the function fun( ) is called for the first time then the complier will allocate space of 2 bytes for 'x' and will initialize it with the value 1.

The value of 'x' i.e. 1 will be printed and then it will be incremented to 2. The function call will then be finished. However, the 'x' **will not be de-allocated** from the memory because its storage class is **static**. The control will then return back to the main( ).

The function fun( ) will be called again. The function will not re-initialize the value of 'x' by 1. This is because the concept is, **initialization while declaration** whereas the **'x' is already declared** and it is there in the memory. Hence the existing value of 'x' i.e. 2 will be printed and then the 'x' will be incremented to 3 and the function call gets finished. The process will be repeated and the output will be,

1 2 3 4 5

Points to Note:

1. In both the above programs, the storage class of the variable 'i' was auto by default.
2. Both the auto and static storage class variables are stored in primary memory of computer (RAM).
3. The life span of the auto variable is from the time at which the function containing that variable is called, to the time at which that function call gets finished.
4. The life span of the static variable is from the time at which the function containing that variable is called to the time at which the main function i.e. the program execution gets finished.

7.3.2 register storage class –

- When the storage class of the variable is given as register then it will be stored in one of the general purpose registers within the processor. If none of the processor's registers is free then the variable will be stored in primary memory just like auto and static variables.
- Consider the following example.

```
register int x;
```

If any register within the processor is free then the variable 'x' will be stored within that register otherwise it will be stored in primary memory and then it will just act as variable with auto storage class. The register storage class will be used with the variables that require frequent calculations. Since these variables are stored within one of the registers within the processor only, those will be accessed much faster and thus the execution time will reduce.

7.3.3 extern storage class –

- Consider the following example.

```
#include<stdio.h>
int x=123;
void main()
{
    int y=45;
    printf("%d %d\n", x, y);
}
```

- In the above program, the 'y' is a **local variable** of main( ) and its storage class is **auto by default**. The variable 'x' is **external variable** whose storage class is **static by default**. In addition to that the storage class of variable 'x' is extern. This is because it is defined external to the function. The output of the program will be **123 45**.
- Consider the following example.

```
#include<stdio.h>
void main()
{
    int y=45;
    printf("%d %d\n", x, y);
}
int x=123;
```

➢ The above program will result into compilation error indicating that 'x' is the undefined symbol. This is because the 'x' is referred first and defined later. The above error can be removed by writing the program as follows.

```
#include<stdio.h>
void main( )
{
        extern int x;
        int y=45;
        printf("%d %d\n", x, y);
}
int x=123;
```

➢ In the above program, using the statement,

```
extern int x;
```

we inform to the complier that 'x' is an integer variable and its declaration is available somewhere externally. Hence the complier will search 'x' outside the main( ) and will get it and thus the error will be avoided.

Points to note:

1. The storage class of local variable is auto by default. It can be initialized with the required value and if not initialized then it will be having a garbage value.
2. The local variable can be declared as static. It can be initialized with the required value and if not initialized then it will be having zero value by default.
3. The storage class of external variable is compulsorily static. It can be initialized with the required value and if not initialized then it will be having a zero value by default.

CWP-89: Write the output of the following program.

Ans: Refer your notebook for the program.

7.4 Scope of Identifiers / Scope of Variables –

➢ Consider the following example.

```
#include<stdio.h>
int x;
void main()
{
      int y;
      extern int q;
      void abc(), def(), pqr();
      .....
      .....
      .....
}
```

```
int z;
void abc()
{
     int p;
     ........
}
int q;
void def()
{
     int r;
}
void pqr()
{
     int s;
}
```

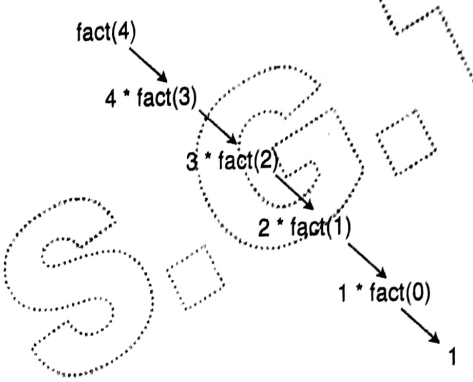
The following table shows the details of all possible variables used in the above program.

Variable name	Storage class and type	Scope
x	extern and static	main( ), abc( ), def( ), pqr( )
y	auto and local	main( )
z	extern and static	abc( ), def( ), pqr( )
p	auto and local	abc( )
q	extern and static	def( ), pqr( ), main( )
r	auto and local	def( )
s	auto and local	pqr( )

- The scope of the variable or the identifier is nothing but the area in which that variable or identifier can be used.
- The scope of the extern variable is from its point of definition till the physical end of the file.
- The scope of the local variable is nothing but the function in which the variable is declared.
- In the above program the scope of the external variable 'x' is nothing but all the functions in the program. It means that the 'x' can be recognized and hence can be used in all the functions.
- The scope of the external variable 'z' is nothing but abc( ), def( ), and pqr( ). Thus it can be used in all these functions but it can not be used in main( ).
- The scope of the external variable 'q' is nothing but def( ) and pqr( ). However, the variable 'q' will also have the scope in main( ) due to the declaration **extern int q;** in the main function. The 'q' can not be used in abc( ).
- The variables **y**, **p**, **r** and **s** are all local variables and hence their scope will be only within the functions in which they are declared. Hence the scope of **y**, **p**, **r** and **s** is within the functions main( ), abc( ), def( ) and pqr( ) respectively.

7.5 Recursion –

- Recursion is the process in which the problem is specified in terms of itself.
- To implement recursion, the function should call itself. The function calling itself is called as recursive function.
- Every invoke of the recursive function will give the partial solution and will call the next invoke of itself to get the further solution.
- There must be some condition by which the recursion process will be stopped, otherwise it can lead to an infinite process.
- When recursion is used then all the partial solutions must be combined to obtain the final solution.
- For e.g. factorial of 4 can be described as follows.



CWP-90: Write a recursive function that accepts one integer number and calculates and returns its factorial. Write a program that will accept the values of n and r from user and will print the value of ${}^{n}C_{r}$ using above function.

Ans: Refer your notebook for the program.

CWP-91: Write a recursive function that accepts one positive integer say x and it should calculate and return sum of all the positive integers from 1 to x. Write a suitable main function that will read one positive integer say n and will print sum of all the positive integers from 1 to n using the above recursive function.

Ans: Refer your notebook for the program.

CWP-92: Write a recursive function that accepts one positive integer and will calculate and return sum of all the digits present it. Write suitable main function.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int n;
     int sum(int);
     clrscr();
     printf("Enter one positive value=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
          printf("Answer=%d\n",sum(n));
     getch();
     return 0;
}
int sum(int x)
{
     if(x>0)
          return x%10 + sum(x/10);
     else
          return 0;
}
```

CWP-93: Write a recursive function that accepts one floating point value x and a non negative integer y. The function should calculate and return x^y. Write suitable main function

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     float x;
     int y;
     float power(float, int);
     clrscr();
     printf("Enter one float value=");
     scanf("%f",&x);
     printf("Enter one non negative integer=");
     scanf("%d",&y);
     if(y<0) printf("Invalid data\n");
     else
          if(x==0)
               if(y==0)
                    printf("Answer not defined\n");
               else
                    printf("Answer=0\n");
          else
               printf("Answer=%f\n",power(x,y));
     getch();
     return 0;
}
float power (float x, int y)
{
     if(y>0)
          return x*power(x,y-1);
     else
          return 1;
}
```

CWP-94: Write a recursive function that accepts two positive integers and calculates and returns their GCD using Euclid's algorithm. Write suitable main function. The Euclid's algorithm to calculate GCD of two numbers is given as follows.

	=	GCD (n, m)	if m < n
GCD (m,n)	=	m	if n = 0
	=	GCD (n, m%n)	otherwise

Ans: Refer your notebook for the program.

CWP-95: Recursive function called as Ackerman's function is popular among the lectures of computer science and that function is as follows.

	=	n + 1	if m = 0
ACK (m,n)	=	ACK (m – 1, 1)	if n = 0 and m > 0
	=	ACK (m – 1, ACK (m, n – 1)	otherwise

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int x,y;
     int ack(int,int);
     clrscr();
     printf("Enter two non negative integers=");
     scanf("%d %d",&x,&y);
     if (x<0 || y<0)
          printf("Invalid data\n");
     else
          printf("Answer=%d\n",ack(x,y));
     getch();
     return 0;
}
int ack(int m,int n)
{
     if(m==0)
          return n+1;
     else
     if(n==0 && m>0)
          return ack(m-1,1);
     else
          return ack(m-1,ack(m,n-1));
}
```

CWP-96: Write a recursive function that will accept one integer i and will calculate and return i^{th} term of the Fibonaccii series. Write a main function that will accept one positive integer say n and will print first n terms of the Fibonaccii series and will also print sum of all these terms.

Ans: Refer your notebook for the program.

CWP-97: Write the output of the following program.

Ans: Refer your notebook for the program and its output.

CWP-98: Write a recursive function that accepts one positive integer in decimal and will print the corresponding binary equivalent. Write a suitable main function that will accept positive integer from user and will print its binary equivalent with the help of recursive function.

Ans: Refer your notebook for the program.

CWP-99: Write a recursive function that accepts one positive integer and print the digits in the reverse order. Write suitable main function.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int n;
     void reverse(int);
     clrscr();
     printf("Enter one positive integer=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
          reverse(n);
     getch();
          return 0;
}
void reverse(int x)
{
     if(x>0)
     {
          printf("%d",x%10);
          reverse(x/10);
     }
}
```

CWP-100: Any positive integers converges to one if the following logic is applied. If the number is even then divide it by 2 and if the number is odd multiply ~~is odd multiply~~ it by 3 and add 1 into it.

Write a recursive function that will accept one positive integer and will print all the intermediate answers from that number upto 1. Write a suitable main function.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int n;
     void fun(int);
     clrscr();
     printf("Enter one positive integer=");
     scanf("%d",&n);
     if(n<=0) printf("Invalid data\n");
     else
          fun(n);
     getch();
     return 0;
}
void fun(int x)
{
     printf("%d ",x);
     if(x>1)
          if(x%2==1)
          fun(3*x+1);
          else
          fun(x/2);
}
```

8. ADDITIONAL C DATA TYPES

8.1 Arrays – Single and Multidimensional –

- Array is nothing but group of elements where all the elements are or same type. The two broad categories of the arrays are,
  a. One dimensional array
  b. Multi-dimensional array

One dimensional Array –

Consider the following example.

```
          7
          -4
a    =    0
          8
          19
```

- In the above example, we have formed group of five integers and the **a** acts as group name. To access individual elements of this group, we have to specify subscript along with the array name.
- The **subscript** is also called as **index**. The elements in the above array are accessed as a[0], a[1], a[2], a[3] and a[4]. Thus the subscript values will range from 0 to 4.
- To access any element in the array only one subscript is sufficient and hence the array a is called as one dimensional array.

Declaration of one dimensional array –

The general syntax of declaring one dimensional array is s follows.

```
datatype arrayname [size]
```

For e.g.

```
int a[5];
```

Using the above declaration we declare to the compiler that **a** is an array of 5 elements and all the elements are of type int. In response to this declaration, the compiler will allocate a space of 10 successive bytes in memory. The subscript values will range from 0 to 4 and the array elements will be accessed as a[0], a[1], a[2], a[3] and a[4]. In general, if one dimensional array is of size 'n' then the subscript values will range from 0 to n – 1.

➢ **Initialization while declaration of one dimensional array –**

The array elements can be initialized at the time of declaration. Consider the following examples.

```
int a[5] = {10, 20, 30, 40, 50};
int b[5] = {10, 20, 30};
int c[5] = {0, 50, 60};
int d[5];
static int e[5];
int f [ ] = {10, 20, 30};
```

Due to the above declarations the array elements will be initialized as follows.

a

0	1	2	3	4
10	20	30	40	50

d

0	1	2	3	4
G1	G2	G3	G4	G5

b

0	1	2	3	4
10	20	30	0	0

e

0	1	2	3	4
0	0	0	0	0

c

0	1	2	3	4
0	50	60	0	0

f

0	1	2
10	20	30

Two Dimensional Array –

Consider the following example.

b

	0	1	2	3
0	12	-7	4	8
1	23	31	19	30
2	14	2	9	28

In the above example, we have a group of 12 integers in the tabular form of 3 rows and 4 columns and the group name is **b**. In the above example if we want to access the element 19 then it will be accessed as **b[1][2]**. Since we specify two subscripts to access the particular element of the array it is called as two dimensional array.

➢ **Declaration of two dimensional array –**

The general syntax of declaring two-dimensional array is as follows.

datatype arrayname [row_size][column_size]

For e.g.

int b[3][4];

- Due to the above declaration we declare to the complier that 'b' is a two dimensional array of Integers of 3 rows and 4 columns. In response to this declaration the complier will allocate a space of 24 successive bytes in memory.
- The array elements will be stored in the memory row-wise. **The first subscript is always a row subscript and the second subscript is always column subscript.**
- In the above example the values of row subscript will range from 0 to 2 and the values of column subscript will range from 0 to 3.
- In general if the two dimensional array is of **m** rows and **n** columns then the values of row subscript will vary from 0 to m – 1 and the values of column subscript will vary from 0 to n – 1.

➢ **Initialization with declaration of two dimensional array –**

Consider the following examples.

```
e.g. 1    int a[3][4] = {
                          {10, 20, 30, 40},
                          {0, 50, 60},
                          {70}
                        };
e.g. 2    int b[3][4] = {
                          {10, 20, 30, 40},
                          {0, 50}
                        };
e.g. 3    int c[3][4] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120};
e.g. 4    static int d[3][4];
e.g. 5    int e[3][4];
```

Due to the above declarations the array elements will be initialized as follows.

b

	0	1	2	3
0	10	20	30	40
1	0	50	60	0
2	70	0	0	0

b

	0	1	2	3
0	10	20	30	40
1	0	50	0	0
2	0	0	0	0

c

	0	1	2	3
0	10	20	30	40
1	50	60	70	80
2	90	100	110	120

d

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

e

	0	1	2	3
0	G00	G01	G02	G03
1	G10	G11	G12	G13
2	G20	G21	G22	G23

➢ **Similarity between simple variable and an individual element of an array –**

All the operations that are possible with the simple variable are all possible with the individual elements of the carry. Consider the following examples.

Simple Variable	Individual element of the array
scanf("%d", &x);	scanf("%d", &a[i]);
x++;	a[i]++;
x = y + z;	a[0] = a[1] + a[2];
if (x>0)	if(a[i] > 0)
---------	---------
else	else
--------	---------

CWP-101: Write a program that accepts one dimensional array of not more than 100 integers. The program should also accept one more element say x. The program should find out and print how many times the element x is present in the given array. (Searching of an element in an array).

Ans: Refer your notebook for the program.

CWP-102: Write a program that accepts one dimensional array of not more than 100 integers. The program should also accept one more element say x. The program should find out and print the subscript values i.e. positions at which the element x is present.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[100];
     int i,x,n,count=0;
     clrscr();

     printf("How many integers (not more than 100)=");
     scanf("%d",&n);

     printf("Enter all elements=");
     for (i=0;i<n;i++)
          scanf("%d",&a[i]);

     printf("Enter element to be searched=");
     scanf("%d",&x);

     for(i=0;i<n;i++)
          if(a[i]==x)
          {
               count++;
               printf("%d is present at position %d\n",x,i);
          }

     if(count==0) printf("%d is not present\n",x);

     getch();
     return 0;
}
```

CWP-103: Write a program that accepts one dimensional array of not more than 100 integers. The program should also accept one more element say x. The program should find out and print whether the element x is present in the given array or not. If x is present then the program should print the position of the first occurrence of the element x in the given array.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a[100];
    int i,x,n;
    clrscr();

    printf("How many integers (not more than 100)=");
    scanf("%d",&n);

    printf("Enter all elements=");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    printf("Enter element to be searched=");
    scanf("%d",&x);

    for(i=0;i<n;i++);
        if(a[i]==x)
        break;

    if(i<n) printf("%d is present at position %d\n",x,i);
    else printf("%d is not present\n",x);

    getch();
    return 0;
}
```

CWP-104: Write a program that accepts one dimensional of not more than 100 integers. The program should sort them in ascending order and print them.

Ans: Refer your notebook for the program.

OR Write a Program to implement linear search (search first occurrence)

bubble sort

CWP-105: Write a program that accepts one dimensional array of not more than 100 integers. The program should find out highest out of them and print it.

Ans: Refer your notebook for the program.

CWP-106: Write a program that accepts one dimensional array of not more than 100 integers. The program should find out highest and lowest out of them and print them.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int a[100];
      int i,n,max,min;
      clrscr();

      printf("How many elements (not more than 100)=");
      scanf("%d",&n);

      printf("Enter elements=");
      for(i=0;i<n;i++)
            scanf("%d",&a[i]);

      max=min=a[0];

      for(i=1;i<n;i++)
            if(a[i]>max)
            max=a[i];
      else
            if(a[i]<min)
                  min=a[i];

      printf("Highest=%d\n",max);
      printf("Lowest=%d\n",min);

      getch();
      return 0;
}
```

CWP-107: Write a program that accepts one dimensional array of not more than 100 integers. The program should change the contents with the following logic.

If a[0] > a[1] Then exchange them.

If a[2] > a[3] Then exchange them.

If a[4] > a[5] Then exchange them and so on.....

The program should print the modified array.

Ans: Refer your notebook for the program.

CWP-108: Write a program that accepts one dimensional array of not more than 100 integers. Assuming that user enter these elements in ascending order the program should modify the same array in such a way that the array elements will be arranged in descending order. The program should print the modified array.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[100];
     int i,n,temp;
     clrscr();
     printf("How many elements (not more than 100)=");
     scanf("%d",&n);
     printf("Enter elements in ascending order=");
     for(i=0;i<n;i++)
          scanf("%d",&a[i]);
     for(i=0;i<n/2;i++)
     {
          temp=a[i];
          a[i]=a[n-1-i];
          a[n-1-i]=temp;
     }
     printf("Descending order\n");
     for(i=0;i<n;i++)
          printf("%d ",a[i]);
     getch();
     return 0;
}
```

CWP-109: Write a program that accepts one dimensional array of not more than 100 integers. The program should rotate the contents of the array and should print the modified array.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[100];
     int i,n,temp;
     clrscr();
     printf("How many elements (not more than 100)=");
     scanf("%d",&n);
     printf("Enter elements=");
     for(i=0;i<n;i++)
          scanf("%d",&a[i]);
     temp=a[n-1];
     for(i=n-2;i>=0;i--)
          a[i+1]=a[i];
     a[0]=temp;
     printf("Modified array\n");
     for(i=0;i<n;i++)
          printf("%d ",a[i]);
     getch();
     return 0;
}
```

CWP-110: Write a program that accepts one dimensional array of not more than 50 floating point numbers. The program should calculate and print their mean, variance and standard deviation.

$$\text{Mean} = \frac{\sum_{i=1}^{n} X_i}{n} \qquad \text{Variance} = \frac{\sum_{i=1}^{n} (X_i - \text{mean})^2}{n} \qquad \text{Std.deviation} = \sqrt{\text{variance}}$$

Ans: Refer your notebook for the program.

CWP-111: Write a program that prints Fibonaccii series of n terms where n is not more than 100. The program should also calculate and print sum of all these terms.

Ans: Refer your notebook for the program.

CWP-112: Write a program that accepts the date in **dd/mm/yyyy** format and the program should print day number of the year.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int year[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
     int dd,mm,yy,sum=0,i;
     clrscr();

     printf("Enter data in dd mm yyyy format=");
     scanf("%d %d %d",&dd,&mm,&yy);

     for(i=1;i<=mm-1;i++)
          sum=sum+year[i];
     sum=sum+dd;

     if(mm>2)
          if(yy%400==0 || yy%4==0&&yy%100!=0)
               sum++;

     printf("It is day number %d of the year\n",sum);
     getch();
     return 0;
}
```

CWP-113: Write a program that accept marks of n students of a single subject out of 100. The program should calculate and print number of students in each grade, where the grades are described as follows.

Marks	Grade
0 – 9	0
10 – 19	1
20 – 29	2
.....	...
.....	...
90 – 99	9
100	10

Ans: Refer your notebook for he program.

CWP-114: Write a program that accepts array of not more than 100 integers. The program should also accept element x which is to be inserted into an array and it should also accept position j at which the element x is to be inserted. The program should print the modified array.

Ans: Refer your notebook for he program.

CWP-115: Write a program that accepts array of not more than 100 integers. The program should also accept element x which is to be deleted from an array. Delete only the first occurrence of an element x. The program should print the modified array.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[100];
     int i,j,n,x;
     clrscr();
     printf("How many element (not more than 100)=");
     scanf("%d",&n);

     printf("Enter elemtns=");
     for(i=0;<n;i++)
          scanf("%d",&a[i]);

     printf("Enter the element to be deleted=")
     scanf("%d",&x);

     i=0;
     while(i<n&&a[i]!=x)
          i++;
     if (i<n)
     {
          for (j=i+1;j<n;j++)
               a [j-1]=a[j];

          n--;
          printf("Element deleted\n");
     }
     else
          printf("Element to be delted not found\n");

     printf("Array contents\n");
     for (i=0;i<n;i++)
          printf("%d ",a[i]);
     getch();
     return 0;
}
```

CWP-116: Write a program that accepts array of not more than 100 integers. The program should also accept element x which is to be deleted from an array. Delete all the occurrences of an element x. The program should print the modified array.

Ans:

```
#include<stdio.h>
#inlcude<conio.h>
int main()
{
      int a[100];
      int i,j,n,x,count=0;
      clrscr();
      printf("How many elements (not more than 100)=");
      scanf("%d",&n);

      printf("Enter elements=");
      for(i=0;i<n;i++)
           scanf("%d",&a[i]);
      printf("Enter the element to be deleted=");
      scanf("%d",&x);

      i=0;
      while(i<n)
      {
           while(i<n&&a[i]!=x)
                i++;
           if(i<n)
           {
                for(j=i+1;j<n;j++)
                     a[j-1]=a[j];
                n--;
                count++;
           }
      }

      if(count==0) printf("Element to be deleted is not present\n");

      printf("Array contents\n");
      for(i=0;i<n;i++)
           printf("%d ",a[i]);
      getch();
      return 0;
}
```

CWP-117: Write a program that accepts two matrices of order not more than 10 X 10. Your program should calculate and print their addition and subtraction matrices.

Ans: Refer your notebook for the program.

CWP-118: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should calculate and print sum and average of both the diagonals independently.

Ans: Refer your notebook for the program.

CWP-119: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should calculate and print sum and average of diagonal elements, lower triangular elements and upper triangular elements independently.

Ans: Refer your notebook for the program.

CWP-120: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should accept two row numbers say x and y and should exchange these two rows. The program should print the modified matrix after exchanging these rows.

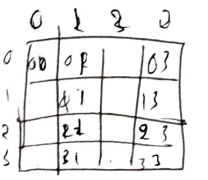
Ans: Refer your notebook for the program.

CWP-121: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should accept two column numbers say x and y and should exchange these two columns. The program should print the modified matrix after exchanging these columns.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[10][10];
     int i,j,m,n,x,y,temp;
     clrscr();
     printf("Enter order of matrix (not more than 10X10)=");
     scanf("%d %d",&m,&n);
     printf("Enter element of matrix=");
     for(i=0;i<m;i++)
     for(j=0;j<n;j++)
          scanf("%d",&a[i][j]);
     printf("Enter column numbers which are to be exchange=");
     scanf("%d %d",&x,&y);
     for(i=0;i<m;i++)
     {
          temp=a[i][x];
          a[i][x]=a[i][y];
          a[i][y]=temp;
     }

     printf("Modified array\n");
     for(i=0;i<m;i++)
     {
          for(j=0;j<n;j++)
               printf("%d",a[i][j]);
          printf("\n");
     }
     getch();
     return 0;
}
```



CWP-122: Write a program that accepts two matrices of order not mote than 10 X 10. Your program should calculate and print their multiplication matrix.

Ans: Refer your notebook for the program.

CWP-123: Write a program that prints the following output.

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
```

Same as CWP 70

Hint:

P[i][j] = 1 when j = 0 or j = i

P[i][j]=P[i−1][j−1]+P[i−1][j] otherwise for all values of j for which j < i

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int p[6][6];
     int i,j;
     clrscr();
     for(i=0;i<6;i++)
     {
          for(j=0;j<=i;j++)
          {
               if(j==0 || j==i) p[i][j]=1;
               else
                    p[i][j]=p[i-1][j-1]+p[i-1][j];
               printf("%d/t",p[i][j])
          }
          printf("\n");
     }
     getch();
     return 0;
}
```

CWP-124: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should obtain its transpose into another array and print it.

Ans: Refer your notebook for the program.

CWP-125: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should obtain its transpose into the same array and print it.

Ans: Refer your notebook for the program.

CWP-126: Write a program that accepts a rectangular matrix of order not more than 10 X 10. Your program should obtain its transpose into another array and print it.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[10][10],b[10][10];
     int i,j,m,n;
     clrscr();

     printf("Enter order of matrix(not more than 10)=");
     scanf("%d %d",&m,&n);

     printf("Enter elements of matrix=");
     for(i=0;i<m;i++)
     for(j=0;j<n;j++)
          scanf("%d",&a[i][j]);

     for(i=0;i<m;i++)
     for(j=0;j<n;j++)
          b[j][i]=a[i][j];

     printf("Transpose matrix\n");
     for(i=0;i<n;i++)
     {
          for(j=0;j<m;j++)
               printf("%d\t",b[i][j]);
          printf("\n");
     }
     getch();
     return 0;
}
```

CWP-127: Write a program that accepts a rectangular matrix of order not mote than 10 X 10. Your program should obtain its transpose into the same array and print it.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int a[10][10];
      int i,j,m,n,max,temp;
      clrscr();
      printf("Enter oreder of matrix(not more than 10)=");
      scanf("%d %d",&m,&n);

      printf("Enter element of matrix=");
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
      if(m>n) max=m;
      else max=n;

      for(i=0;i<max;i++)
      for(j=0;j<max;j++)
      if(i<j)
      {
            temp=a[i][j];
            a[i][j]=a[j][i];
            a[j][i]=temp;
      }
      printf("Transpose matrix\n");
      for(i=0;i<n;i++)
      {
            for(j=0;j<m;j++)
                  printf("%d\t",a[i][j]);
            printf("\n");
      }
      getch();
      return 0;
}
```

CWP-128: Write a program that accepts a square matrix of order not more than 10 X 10. Your program should determine and print whether this matrix is symmetrical or not.

Ans: Refer your notebook for the program.

CWP-129: Write a program that accepts a matrix of order not more than 10 X 10. Your program should print all the elements of the matrix. The program should print sum of all the elements in each row along with that row. It should also print sum of all the element in each column below that column. The program should also print sum of all the element below the row sums.

Ans:

```
#include<stdio.h>
#include<conio.h>
int main()
{
      int a[10][10];
      int i,j,sumr,sumc,sumt,m,n;
      clrscr();
      printf("Enter order of matrix(not more than 10X10)=");
      scanf("%d %d",&m,&n);

      printf("Enter elements=");
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);

      for(i=0;i<m;i++)
      {
            sumr=0;
            for(j=0;j<n;j++)
            {
            printf("%d\t",a[i][j]);
            sumr=sumr+a[i][j];
            }
            printf("%d\n",sumr);
      }
      for(j=0;j<n;j++)
      {
            sumc=0;
            for(i=0;i<m;i++)
                  sumc=sumc+a[i][j];
            printf("%d\t",sumc);
      }
      sumt=0;
      for(i=0;i<m;i++)
      for(j=0;j<n;j++)
            sumt=sumt+a[i][j];
      printf("%d\n",sumt);
      getch();
      return 0;
}
```

8.2 Strings –

CWP-130: Write a program that accepts a string and prints whether the accepted string is palindrome or not.

Ans: Refer your notebook for the program.

CWP-131: Write a program that accepts a string into character array and the program should reverse the string within the same array i.e. the original string should be replaced with its reversed string and the program should print the reversed string.

Ans: Refer your notebook for the program.

CWP-132: Write a program that accepts a string into character array and the program should also accept one more character and your program should find out and print number of occurrences of this character in the given string.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
      char str[80],x;
      int i,n,count=0;
      clrscr();

      printf("Enter a string=");
      scanf("%[^\n]",str);

      fflush(stdin);
      printf("Enter a character to be searched=");
      scanf("%c",&x);

      n=strlen(str);

      for(i=0;i<n;i++)
            if(str[i]==x)
            count++;

      if(count==0) printf("%c is not present\n",x);
      else printf("%c is present %d times\n",x,count);
      getch();
      return 0;
}
```

CWP-133: Write a program that accepts a string into character array and the program should also accepts one more character and your program should find out and print positions of occurrences of the character in the given string.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
      char str[80],x;
      int i,n,count=0;
      clrscr();
      printf("Enter a string=");
      scanf("%[^\n]",str);
      fflush(stdin);
      printf("Enter a character to be searched=");
      scanf("%c",&x);
      n=strlen(str);

      for(i=0;i<n;i++)
            if(str[i]==x)
            {
                  printf("%c is present at position %d\n",x,i);
                  count++;
            }
      if(count==0) printf("%c is not present\n",x);
      getch();
      return 0;
}
```

CWP-134: Write a program that accepts and calculates and prints number of vowels, consonants, digits and other characters present ion that string.

Ans: Refer your notebook for the program.

CWP-135: Write a program that accepts a string and put it into character array. The program should modify the string in such a way that first letter of every word will be printed in capitals.

Ans: Refer your notebook for the program.

CWP-136: Write a program that accepts a string and put it into character array. If the string contains multiple blank space between the words then those multiple blank spaces should be replaced by single blank space.

Ans:

```
#include<iostream.h>
#nclude<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
      char str[80],tempstr[80];
      int i,j,k=1,n;
      clrscr();
      cout<<"Enter a string=";
      gets(str);
      n=strlen(str);
      tempstr[0]=str[0];
      for(i=1;i<n;i++)
            if(    str[i-1]==' '&&str[i]!=' '
                 ||str[i-1]!=' '&&str[i]==' '
                 ||str[i-1]!=' '&&str[i]!=' ')
            {
                 tempstr[k]=str[i];
                 k++;
            }
      tempstr[k]='\0';
      for(i=0;i<=k;i++)
            str[i]=tempstr[i];
      cout<<"Modified string\n";
      puts(str);
      getch();
}
```

CWP-137: Write a program that accepts a string and prints its length without using library function strlen(). OR write a used defined function to calculate length of string

Ans: Refer your notebook for the program.

CWP-138: Write a program that accepts a string and prints number of words present in the given string.

Ans: Refer your notebook for the program.

CWP-139 Write a program that accepts into character array s1 and copies it into another character array s2 without using the library function strcpy(). The program should print the copied string.

Ans: Refer your notebook for the program.

CWP-140 Write a program that accepts first string into character array s1 and the second string into character array s2. The program should concatenate the string s1 with the string s2 without using he library function strcat(). The concatenated string should become available in array s2. The program should print the concatenated string.

Ans: Refer your notebook for the program.

CWP-141 Write a program that accepts main string into character array s1 and the substring into character array s2. The program should find out and print the positions at which the substring is present in the main string.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
     char s1[80],s2[80];
     int i,j,n1,n2,count=0;
     clrscr();
     printf("Enter main string=");
     scanf("%[^\n]",s1);
     fflush(stdin);
     printf("Enter substring string=");
     scanf("%[^\n]",s2);

     n1=strlen(s1);
     n2=strlen(s2);

     for(i=0;i<=n1-n2;i++)
     {
          for(j=0;j<n2;j++)
               if(s1[i+j]!=s2[j])
                    break;
          if(!(j<n2))
          {
               printf("Position=%d\n",i);
               count++;
          }
     }
     if(count==0) printf("Substring is not present in main string\n");
     getch();
     return 0;
}
```

CWP-142: Write a program that accepts a string and print the words in the reverse order. For example,

Input String : Insects Eat Apples

Output String : Apples Eat Insects

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
     char str[80];
     int f,b,i,n;
     clrscr();
     printf("Enter a strings=");
     scanf("%[^\n]",str);
     n=strlen(str);
     f=b=n-1;
     while(f!=-2)
     {
          while(f!=-1&&str[f]!=' ')
               f- -;
          for(i=f+1;i<=b;i++)
               printf("%c",str[i]);
          if(f!=1) printf(" ");
          f=b=f-1;
     }
     getch();
     return 0;
}
```

8.3 Pointers –

8.3.1 Basics of Pointers–

Definition–

The pointer is nothing but an address. The pointers are useful for indirect access. The pointer are useful for the implementation of **call by address.**

Declaration of pointer variable–

The general syntax of declaring pointer variable is as follows

datatype*pointername;

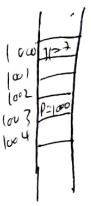
For e.g.

```
int*p;
float*q;
char*r;
```

In the above declaration, the p will act as pointer to integer, the q will ct as pointer to float and the r will act as pointer to char. In other words, the p will act as reference to integer, the q will act as reference to float and the r will act as reference to char. In other words, the p will store address of integer, the q will store address of float and the r will store address of char.

Referencing and De-referencing –

The **address-of operator (&)** is used for **referencing** and the **indirection operator(*)** is used for **de-referencing**. To understand the concept of the referencing and de-referencing let us consider the following example.

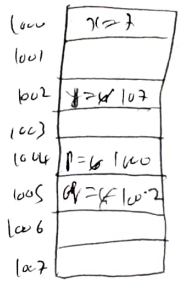



```
#include<stdio.h>
void main()
{
    int x=7;
    int *p;
    p=&x;
    printf("%d is stored at address %x\n",*p,p);
}
```

- In the above program the **p** is declared as pointer to integer i.e. reference to integer.
- Consider the statement, **p**=&x; In the statement, we are using **address-of operator i.e., &**. Due to this statement the address of x is copied into **p**. In other words the **p** will store **address of x** i.e. **p** will act as **pointer to x** i.e. **p** will act as **reference to x.** This process is called as **referencing.**
- In the **printf()** statement we are printing the value of ***p**. The **operator*** is nothing but **indirection operator.** The indirection operator operates on address i.e. reference and gives **value stored at that address**. Since **p** store the **address of x**, the value of ***p** is nothing but the value of **x**. Thus we can refer **x** indirectly as ***p** and hence **indirection operator**. We can also say that by applying the indirection operator on the pointer **p** i.e. reference **p** we are getting the original value of **x**, hence this concept is called as **de-referencing**. Hence, the **operator*** is also called as **de-referencing operator.**

Write the output of the following program.

```
#include<stdio.h>
void main()
{
      int x=7, y;
      int *p, *q;
      p = &x;
      q = &y;
      *q = *p + 100;
      printf("%d\t%d\n",x,*p);
      printf("%d\t%d\n",y,*q);
}
```



Output:

7 7
107 107

8.3.2 Pointer to Pointer–

The general syntax of declaring pointer to pointer is as follows.

datatypepointer name**

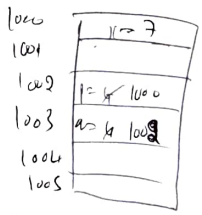
The e.g. will be as follows.

intq;**

In the above example, the **q is pointer to pointer to integer**. In other words, the **q stores address of address of integer.**

Write the output of the following program.

```
#include<stdio.h>
void main()
{
      int x=7;
      int *p;
      int **q;
      p = &x;
      q = &p;
      printf("%d is stored at %u\n",x,p);
      printf("%d is stored at %u\n",*p,p);
      printf("%d is stored at %u\n",**q,*q);
      printf("%u is stored t %u\n",*q,q);
}
```

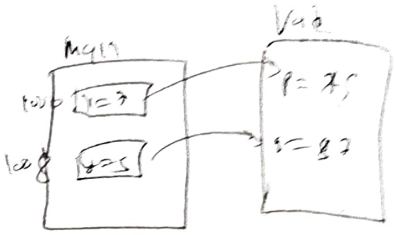


7 is stored at 1000
7 is stored at 1000
7 is stored at 1000
1000 is stored at 1002

Output:

8.3.3 Call by value/Pass by value/Parameter Passing by value-

```
#include<stdio.h>
void main()
{
     int x=7, y=5;
     void swap(int,int);
     printf("Before exchange=");
     printf("x=%d, y=%d\n",x,y);
     swap(x,y);
     printf("After exchange=");
     printf("x=%d, y=%d\n",x,y);
}
void swap (int p, intq)
{
     int temp;
     temp=p;
     p=q;
     q=temp;
}
```



Output:

Before exchange => x=7, y=5

After exchange => x=7, y=5

Explanation:

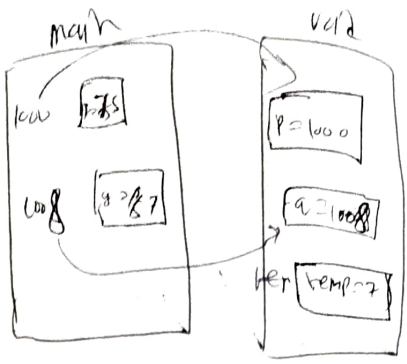
When the swap() is **called**, then the **values** of actual parameters x and y are **passed** and copied into formal parameters i.e. p and q. Hence the concept is referred as **call by value** or **pass by value.**

The actual parameters and formal parameters occupy separate memory location and hence exchange taking place in p and q will not reflect changes in x and y.

The values of actual parameters x and y are copied into formal parameter p and q but the changes in formal parameters p and q are not reflected back into actual parameter x and y. Thus, call by value is useful for one-way communication through parameters.

8.3.4 Call by Address/Call by pointer/Call by Reference/Pass by Address//Pass by Pointer/ Pass by Reference–

```
#include<stdio.h>
void main()
{
      int x=7, y=5;
      void swap(int *,int *);
      printf("Before exchange=");
      printf("x=%d, y=%d\n",x,y);
      swap(&x,&y);
      printf("After exchange=");
      printf("x=%d, y=%d\n",x,y);
}
void swap(int *p, int *q)
{
      int temp;
      temp = *p;
      *p = *q;
      *q = temp;
}
```



Output:

Before swap exchange => x=7, y=5

after exchange => x=5, y=7

Explanation:

When the swap() is **called**, then the **address** of x and y i.e. in other words **reference** of x and y are **passed** as parameters. Hence the concept as **call by reference** or **call by address** or **pass by reference** or **pass by address**.

When the swap() is called, the addresses of x and y are copied into pointer variables named as p and q. The logic within the swap() exchanges the values of variable whichare being pointed by p and q. Thus, with the help of **reference** p and q and by **de-referencing** them, the values of x and y itself are exchanged.

Due to pointer, the function swap() understands that, at which location the exchange is to be made and the exchange made are reflected back into x and y within the main(). Thus, call by address i.e. call by reference is useful for two-way communication.

➢ **Note:** When the *appears in the statement part it is nothing but **operator**. If it is used as unary operator then it is indirection. If it is used binary operator then it is multiplication operator. When the *appears in the declaration part or header then it is indicator, indicating that we are dealing with pointers.

8.3.5 One dimensional Array and Pointers–

One dimensional array name written without subscript is nothing but pointer to the **0th** element of the array or in other words address of the **0th** element of the array.

Consider the following example.

Int a[50];

Note 1- **a[0]** **Is 0th element array a.**

a **Is address of 0th element of array a.**

Note 2-

a	←→	&a[0]
a + 0	←→	&a[0]
a + 1	←→	&a[1]
a + 2	←→	&a[2]
.....		
.....		
a + I	**←→**	**&a[i]**

Note 3–

***(a+ i)** **←→** **a[i]**

Write the output the following program.

```
#include<stdio.h>
void main()
{
     int a[5]={10,20,30,40,50};
     int i,*y;
     y=a;
     for(i=0;i<5;i++)
          printf("%d\t", y[i]);
}
```

Output:

In the above program, y acts as another name of array a.

Write the output of the following program.

```
#include<stdio.h>
void main()
{
     int a[5]={10,20,30,40,50};
     int i;
     for(i=0;i<5;i++)
          printf("%d\n",i[a]);
}
```

Output:

10
20
30
40
50

8.3.6 Two dimensional array and pointers–

Two dimensional array name writtern without subscript is nothing but the pointer to pointer to the element in the 0^{th} row and 0^{th} column or in other words address of address of the element in the 0^{th} row and 0^{th} column.

Consider the following example.

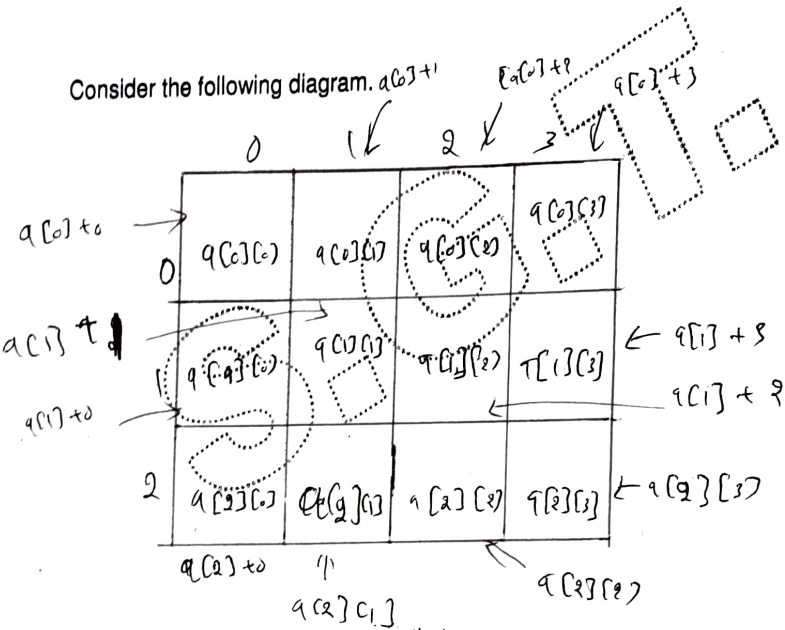
```
Int a[3][4];
```

Note 1– **a[0][0]** **is element in the 0^{th} row and 0^{th} column of array a.**

a[0] **is address of element in the 0^{th} row and 0^{th} column of array a.**

a **is address of address of element in the .0^{th} row and 0^{th} column of array a.**

Consider the following diagram.



From the above diagram we can say that,

Note 2 –

&a[i][j] ⟷ ***(a+i)+j**

&[0][j] ⟷ ***(*(a+i)+j)**

q[i][j] ↔ value of element at ith row and jth column

q[i]+j ↔ &q[i][j]

*(a+i)+j ↔ &q[i][j]

((a+i)+j) ↔ &q[i][j]

8.3.7 Array of pointers –

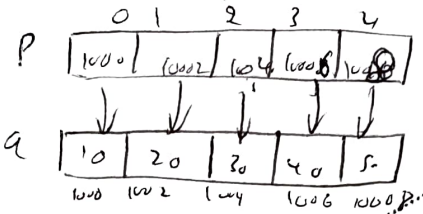
The general syntax of declaring **one dimensional array of pointers** is as follows

datatype*arrayname[size]

For e.g.

int*p[5];

In the above example, p is an array of 5 pointers and all the pointers are pointer to integer.



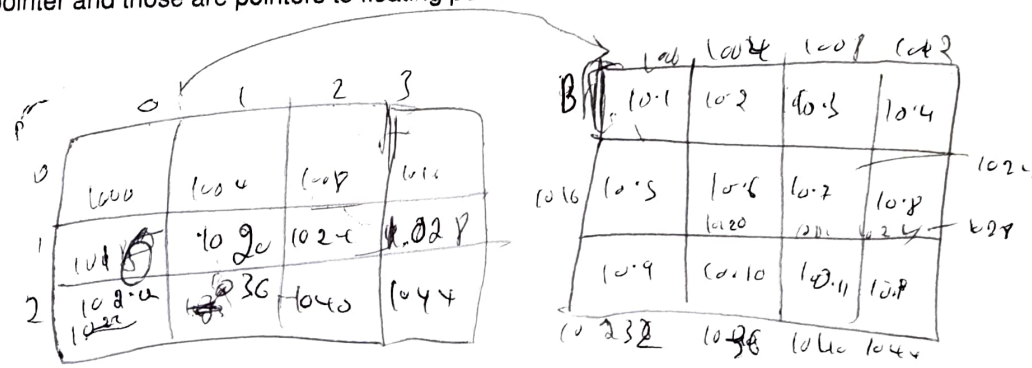
The general syntax of declaring **two dimensional array of pointers** is as follows.

datatype*arrayname[rowsize][columnsize]

For e.g.

float*q[3][4];

In the above example, q is an array of 3 rows and 4 columns and all the 12 elements are pointer and those are pointers to floating point data.



CWP-143: Write the output of the following program.

Ans. Refer your notebook for the program.

8.3.8 One Dimensional Array as parameter to function–

NOTE a[0] → 0th element of an array a.

a → address of / pointer to the 0th element of an array a.

CWP-144: Write a function that accepts one dimensional array of not more than 100 integers and it should also accept the element to be searched. The function should calculate and return the value of how many times the value is present in the given array. Write a suitable main function that will search the element in the array using the above function.

Ans. Refer your notebook for the program.

CWP-145: Write a function that function that accepts one dimensional array of not more than 100 integers and it should also accept the element to be searched. The function should calculate and return the value of how many times the value is present in the given array. Write a suitable main function that will search the element in the array using the above function. Use pointer notation.

Ans. Refer your notebook for the program.

CWP-146 Write a function that function that accepts one dimensional array of not more than 100 integer and it should also accept the element to be searched. The function should return the position of the first occurrence of that element in the array. If the element is not present then it should return –1. Write a suitable main function that will search the element in the array using the above function.

Ans.

```
#include<stdio.h
#include<conio.h>
int main()
{
     int a[100];
     int i,n,x,y;
     int search(int [],int,int);
     clrscr();

     printf("How many elements(not more than 100)=");
     scanf("%d",&n);

     printf("Enter elements=");
     for(i=0;i<n;i++)
          scanf("%d",&a[i]);

     printf("Enter the element to be searched=");
     scanf("%d",&x);

     y=search(a,n,x);

     if(y==-1) printf("%d is not present\n",x);
     else printf("%d is present at position %d\n",x,y);

     getch();
     return 0;
}
int search(int p[],int  n,int x)
{
     int i;
     for (i=0;i<n;i++)
          if(p[i]==x)
               break;
     if(i<n) return i;
     else return -1;
}
```

CWP-147: Write a function that accepts one dimensional array of not more than 100 integer and it should also accepts the element be searched. The function should return the position of the first occurrence of that element in the array. If the element is not present then it should return -1. Write a suitable main function that will search the element in the array using the above function. Use pointer notation.

Ans.

```
#include<stidio.h>
#include<conio.h>
int main()
{
     int a[100];
     int i,n,x,y;
     int search(int *,int,int);
     clrscr();

     printf("How many elements(not more than 100)=");
     scanf("%d",&n);

     printf("Enter elements=");
     for(i=0;i<n;i++)
     scanf("%d",(a+i));

     printf("Enter the element to be searched=");
     scanf("%d",&x);

     y=search(a,n,x);

     if(y==-1) printf("%d is not present\n",x);
     else printf("%d is present at position %d\n",x,y);
     getch();
     return 0;
}
int search(int *p,int n,int x)
{
     int i;
     for(i=0;i<n;i++)
          if(*(p+i)==x)
               break;
     if(i<n) return i;
     else return -1;
}
```

CWP-148: Write a function that accepts string and returns the answer indicating whether that string is palindrome or not. Write suitable main function.

Ans. Refer your notebook for the program.

CWP-149: Write function that accepts floating point array and the function should also accept the option. If the option is 1 then the array should be sorted in ascending order and if the option is 2 then the array should be sorted in descending order. Write suitable main function.

Ans. Refer your notebook for the program.

CWP-150: Write a function that accepts floating point array and calculate and returns their mean. Also write a function that will accept floating point array and calculates and return their standard deviation. Write suitable main function.

Ans.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
      float a[100];
      int i,n;
      float mean(float [],int);
      float sd(float [],int);
      clrscr();
      printf("How many elements=");
      scanf("%d",&n);
      printf("Enter elements=");
      for(i=0;i<n;i++)
            scanf("%f",&a[i]);
      printf("Mean=%f\n",mean(a,n));
      printf("Std. deviation=%f\n",sd(a, n));
      getch();
      return 0;
}
float mean(float p[],int n)
{
      int i;
      float sum=0;
      for(i=0;i<n;i++)
            sum=sum+p[i];
      return sum\n;
}
float sd(float p[],int n)
{
      int i;
      float sum=0,m;
      m=mean(p,n);
      for(i=0;i<n;i++)
            sum=sum+(p[i]-m)*(p[i]-m);
      return sqrt(sum/n);
}
```

8.3.9 Two dimensional Array as Parameter to function –

CWP-151: Write a program to check the property, $(A \times B)^T = B^T X A^T$. The program should have following functions

(1) A function to read a matrix
(2) A function to print a matrix
(3) A function to multiply two matrices
(4) A function to find out transpose of a matrix
(5) A function to check equality of two matrices.
(6) The main function

Ans: Refer your notebook for the program

CWP-152: Write a menu driven program that will provide options for addition, subtraction, multiplication of matrices and transpose of a matrix. The program should give facility for repeated execution till user enters the option quit. Write the program with the help of modular programming approach.

Ans. use functions

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a[10][10],b[10][10],c[10][10];
    int m1,n1,m2,n2,option;
    void readmat(int [][10],int,int);
    void printmat(int [][10],int,int);
    volid addmat(int [][10],int [][10],int [][10],int,int);
    void submat(int [][10],int [][10],int [][10],int,int);
    void multmat(int [][10],int [][10],int [][10],int,int int);
    void transpose (int [][10],int [][10],int,int);
    do
    {
        clrscr();
        printf("1:Add,2:Subtract,3:Multiply,4:Transpose, 5:Quit\n);
        printf("Enter option=");
        scanf("%d",&option);
        if(option>=1&&option<=3)
        {
            printf("Enter order of first matrix=");
            scanf("%d %d",&m1,&n1);

            printf("Enter order of second matrix=");
            scanf("%d %d",&m2,&n2);

            printf("Enter elements of first matrix=");
            readmat(a,m1,n1);

            printf("Enter elements of second matrix=");
            readmat(b,m2,n2);
            switch(option)
            {
                case 1:if(m1==m2&&n1==n2)
                       {
                           addmat(a,b,c,m1,n1);
                           printf("Addition\n");
                           printmat(c,m1,n1);
                       }
                       else printf("Order mismatch\n");
                       break;
                case 2:if(m1==m2&&n1==n2)
                       {
                           submat(a,b,c,m1,n1);
                           printf("Subtraction\n");
                           printmat(c,m1,n1);
                       }
                       else printf("Order mismatch\n");
                       break;
                case 3:if(n1==m2)
                       {
                           multmat(a,b,c,m1,n2,n1);
                           printf("Multiplication\n");
                           printmat(c,m1,n2);
                       }
                       else printf("Order mismatch\n");
                       break;

        }
        getch();
    }
```

```
            else
            if(option==4)
            {
                printf("Enter order of matrix=");
                scnaf("%d %d",&m1,&n1);
                printf("Enter elements=");
                readmat(a,m1,n1);
                transpose(a,c,m1,n1);
                printf("Transpose\n");
                printmat(c,n1,m1);
                getch();
            }
        }
        while(option!=5);
        return 0;
}
void readmat(int p[][10],int x,int y)
{
      int i,j;
      for(i=0;i<x;i++)
      for(j=0;j<y;j++)
           scanf("%d",&p[i][j]);
}
void printmat(int p[][10],int x,int y)]
{
      int i,j;
      for(i=0;i<x;i++)
      {
           for(j=0;j<y;j++)
                printf("%d=\t",p[i][j]);
           printf("\n");
      }
}
void addmat(int p[][10],int q[][10],int r[][10],int x,int y)
{
      int i,j;
      for(i=0;i<x;i++)
      for(j=0;j<y;j++)
           r[i][j]=p[i][j]+q[i][j];
}
void submat(int p[][10],int q[][10],int r[][10],int x,int  y)
{
      int i,j;
      for(i=0;i<x;i++)
      for(j=0;j<y;j++)
           r[i][j]=p[i][j]-q[i][j];
}
void multmat(int p[][10],int q[][10],int r[][10],int x,int y, int z)
{
      int i,j,k;
      for(i=0;i<x;i++)
      for(j=0;j<y;j++)
      {
           r[i][j]=0;
           for(k=0;k<z;k++)
                r[i][j]=r[i][j]+p[i][k]*q[k][j];
      }
}
void transpose(int p[][10],int q[][10],int x,int y)
{
      int i,j;
      for(i=0;i<x;i++)
      for(j=0;j<y;j++)
           q[j][i]=p[i][j];
}
```

8.3.10 Function Returning a Pointer/Address –

CWP-153: Write a function that will accept one dimensional array of integers and will return the address of the highest. Write suitable main function that will read array of integers from user and will print the highest out of it using the above function.

Ans.

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[100],n,i;
     int*highest(int [],int);
     clrscr();
     printf("How many elements (not more than 100)=");
     scanf("%d",&n);
     printf("Enter elements=");
     for(i=0;i<n;i++)
          scanf("%d",&a[i]);

     printf("Highest=%d\n",*highest(a,n));
     getch();
     return 0;
}
int* highest(int p[],int n)
{
     int i,max;
     int *q;

     max=p[0];
     q=&p[0];

     for(i=1;i<n;i++)
     if(p[i]>max)
     {
          *max=p[i];
          q=&p[i];
     }
     return q;
}
```

8.3.11 Pointer to Function –

CWP-154: Write the output of the following program.

Ans.

```
#include<stdio.h>
#include<conio.h>
int main()
{
     int add(int,int);
     int mul(int,int);
     int sub(int,int);
     int (*pf)(int,int);
     int option,x,y;
     clrscr();
     printf("Enter two integers=");
     scanf("%d %d",&x,&y);
     printf("Enter option(1:Add,2:Subtract,3:Multiply)=");
     scanf("%d",&option)
     switch(option)
     {
          case 1:pf=add;break;
          case 2:pf=sub;break;
          case 3:pf=mul;break;
     }
     if(option>=1&&option<=3) printf("Answer=%d\n",(*pf)(x,y));
     else printf("Invalid option\n");
     getch();
     return 0;
}
int add(int p,int q)
{
     return p+q;
}
int sub(int p,int q)
{
     return p-q;
}
int mul(int p,int q)
{
     return p*q;
}
```

8.3.12 Miscellaneous programs –

CWP-155: Write the output of the following program.

Ans. Refer your notebook for the program and the output.

8.4 Structures –

8.4.1 Introduction to structures –

- The C allows us to create our own data type (i.e. user defined data type). Due to this facility, programmer can create his or her own data type for a specific application.
- A structure is nothing but group of data elements, which can be same type as well as different types resulting into meaningful data element satisfying real life requirements.

Defining a structure data type –

The general syntax of declaring structure data type is follows.

```
struct datatype
{
        datatype member1;
        datatype member2;
        .....
        .....
        datatype member n;
};
```

As the general syntax indicates, the structure data type is made up of members. These members can be of different data types.

Example –

```
struct student
{
        int roll_no;
        float percent;
        char name[40];
};
```

In the above example, the **student** is define as structure datatype with members, roll_no of type integer, percent of type float and name which is array of 40 characters. The **structure data type** is also called as **structure tag.**

Declaring variable of structure data type –

The general syntax of declaring a structure variable is as follow.

```
struct datatype variable name
```

Example –

```
struct student s1;
```

Due to the above example, the **variable s1** of type **struct student** is declared. The compiler will allocate a space of 46 bytes to store the information of s1.

The structure data type and the structure variable can be declared together or separately.

Method 1–

```
struct student
{
        int roll_no;
        float precent;
        char name[40];
}s1;
```

Method 2–

```
        struct student
{
        int roll_no;
                float precent;
        char name[40];
};
struct student s1;
```

The net effect of both the above methods is same.

Accessing Individual Members of the Structure –

- When we define a structure data type and its variable, then it become necessary to deal with every individual member to the structure.
- The general syntax of accessing the structure members is as follows.

structure variable name.member name

Example –

```
struct student
{
int roll_no;
float percent;
char name[40];
};
struct student s1;
.....
.....
s1.roll_no=12;
s1.percentage=86.33;
s1.name="xyz";
.....
.....
```

As the example indicates, using three assignment statement, we have assigned information to every individual member of a structure.

Assignment of Complete Structure Variable :-

Consider the following example.

Example :-

```
struct student
{
        int roll_no;
        float percent;
        char name[40];
};
struct student s1, s2;
.....
.....
s1.roll_no=2;
s1.percent=87.67
s1.name="abc";
s2=s1;
.....
.....
```

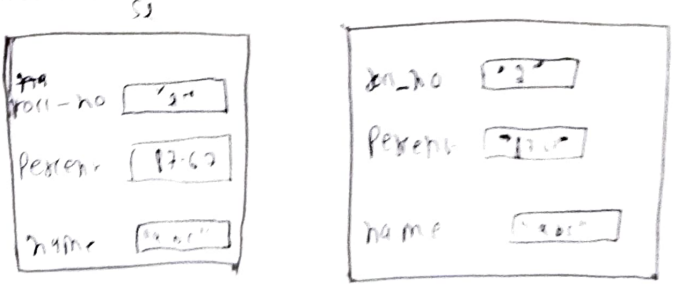
Due to the statement,

```
s2=s1;
```

the values of all the members of s1 are copied into corresponding members of s2.
The effect of,

```
s2=s1;
```

can be shown as follows.



The statement s2=s1; carry out the job of three individual assignment statements as shown below.

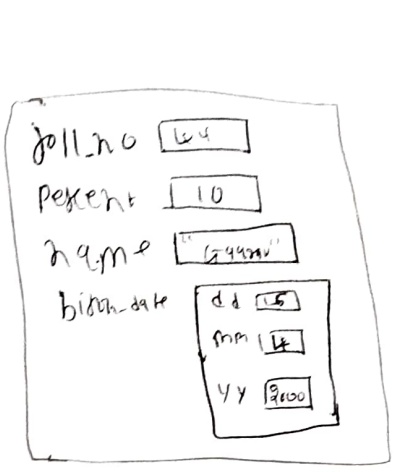
```
s2.roll_no=s1.roll_no;
s2.percent=s1.percent;
s2.name=s1.name;
```

8.4.2 Nested Structures / Embedded Structures

Structure within Structure / Structure as members of Structure –

It is possible to define a structure as a member of the other structure resulting into Nested structure. Consider the following example.

```
struct date
{
        int dd, mm, yy;
};
struct student
{
        into roll_no;
        float percent;
        char name[40];
        struct date birth_date;
};
struct student s1;
```



In this example, we have defined structure **data** type date with three members **dd** **mm** and **yy**. The **student** is also defined as structure data type with following members.

roll_no of type int
percent of type float
name as array of 40 characters
birth_date of type date

- It means that, member birthdate of structure itself as another structure.
- The member of s1 can be accessed as follows

s1.roll _no
s1.percent
s1.name
s1.birth_date

If we want to access individual member of birth_date,
then it will be done as follows

s1.birth_date.dd
s1.birth_date.mm
s1.birth_date.yy

8.4.3 Array of Stru

- We have de
information o
- Consider the

```
struct stude
{
        int ro
        char
        float
};
struct stud
```

In the abov
struct stud
The memb

```
s[i]
s[i
s[i
```

CWP-15
print wh
today's

Ans: Re

CWP-1
student
print ro

Ans: F

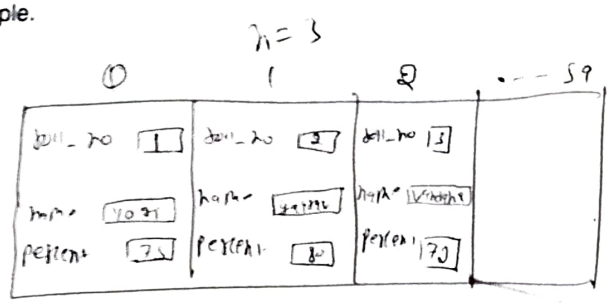
CWP-
book

The
or
of
av

8.4.3 Array of Structures –

- We have defined a structure data type as **student**. When we want to deal with information of number of students, then we can define array of struct student.
- Consider the following example.

```
struct student
{
        int roll_no;
        char name[40];
        float percent;
};
struct student s[60];
```



In the above example, s is defined as array of 60 elements and every element is of type struct student. In other words, s is array of 60 students.

The members of the i^{th} student can be accessed as follows

```
s[i].roll_no
s[i].name
s[i].percent
```

CWP-156: Write a program that today's date and any other date. The program should print whether the other date has been already passed or yet to come or whether it is today's date only.

Ans: Refer your notebook for the program

CWP-157: Write a program that reads roll_no and marks of PCM of not more than 60 students. Your program should calculate average of PCM of all students and it should print roll_no and average of all students.

Ans: Refer your notebook for the program

CWP-158: Write a program that reads information of into than 100 books, where the book details are as follows.

(a) book_id
(b) author_name
(c) title of the book
(d) cost of the book

The program should calculate sum of the cost of the all the books whose cost is Rs. 300 or more individually and also their average cost. The program should also calculate sum of the cost of all the books whose cost is between Rs. 100 and Rs. 200 and also their average cost.

Ans:

```
#include<stdio.h>
#include<conio.h>
struct book
{
      int id;
      char author_name[80];
      char title[80];
      int cost;
};
int main()
{
      struct book b[100];
      int i,n,count1=0,count2=0;
      unsigned long int sum1=0,sum2=0;
      clrscr();

      printf("How many books (not more than 100)=");
      scanf("%d",&n);

      printf("Enter information of all books\n");
      for(i=0;i<n;i++)
      {
            printf("Id=");
            scanf("%d",&b[i].id);
            fflush(stdin);
            prinf("Author name=");
            gets(b[i].author_name);
            fflush(stdin);
            printf("Title=");
            gets(b[i].title);
            printf("Cost=");
            scanf("%d",&b[i].cost);
      }

      for(i=0;i<n;i++)
      if(b[i].cost>=300)
      {
            count1++;
            sum1=sum1+b[i].cost;
      }
      else if(b[i].cost>=100&&b[i].cost<=200)
            {
                  count2++;
                  sum2=sum2+b[i].cost;
            }
prinf("sum of cost of books whose individual cost is 300 or above=%lu\n",sum1);
if(count1!=0)
printf("Avg. cost of above books is=%.2f\n", (float)(sum1)/count1);
printf("Sum of cost of books whose indivi. cost is between 100 and 200=%d\n"1,sum2);
if (count2!=0)
printf("Avg. cost of above books is=%.2f\n", (float)(sum2)/count2);
getch();
return 0;
}
```

CWP-159: Write a program to read names, roll no. and total marks of a class of 50 students. Arrange the data in the descending order of total marks and print the output with the proper headings

Ans: Refer your notebook for the program

CWP-160: Write a program to implement structure for following fields.

Name
roll_no
total_marks
date_of_birth

The program should provide facility to sort the list (array of structures) on name or roll_no or total marks or date of birth depending on the option entered by user.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct date
{
     int dd,mm,yy;
};
struct student
{
     int roll_no;
     char name[40];
     int total;
     struct date dob;
};
int main()
{
     struct student s[50];
     int i,j,n,option;
     void sort1(struct student[],int);
     void sort2(struct student[],int);
     void sort3(sturct student[],int);
     void sort4(struct student[],int);
     clrscr();
     printf("How many students(not more than 50)=");
     scanf("%d",&n);
     printf("Enter information of all\n");
     for( i=0;i<n;i++)
     {
          printf("Roll no=");
          scanf("%d",&s[i].roll_no);
          fflush(stdin);
          printf("Name=");
          gets(s[i].name);
          printf("Total=");
          scanf("%d",&s[i].total);
          printf("Birthdate=");
          scanf("%d %d %d", &s[i].dob.dd,&s[i].dob.mm,&s[i].dob.yy);
     }
```

Write program to read names, roll no and total marks of 50 students
The program should given an option to the user to arrange data in ascending order of roll no or ascending order of name or descending order of marks and sort accordingly
the program should display with proper headings
use top down modular programming

```
        do
        {
           clrscr();
           printf("Sort on\n1:Roll_no\n2:Name\n3:Total\n4:Birthdate\n5:Quit\n");
           printf("Enter you option=");
           scanf("%d",&option);
           switch(option)
           {
              case 1:sort1(s,n);break;
              case 2:sort2(s,n);break;
              case 3:sort3(s,n);break;
              case 4:sort4(s,n);break;
           }
           if(option>=1&&option<=4)
           {
               printf("Roll_no Name                    Total Birthdate\n");
               for(i=0;i<n;i++)
               {
                   printf("%7d %s",s[i].roll_no,s[i].name);
                   for(j=1;j<=40-strlen(s[i].name);j++)
                        printf(" ");
                   printf("%5d %2d/%2d/%4d\n",

                   s[i].total,s[i].dob.dd,s[i].dob.mm,s[i].dob.yy);
               }
               getch();
           }
        }
        while(option!=5);
        return 0;
}
void sort1(struct student p[],int n)
{
        int i,j;
        struct student temp;
        for(i=n-1;i>0;i- -)
        for(j=0;j<i;j++)
        if(p[j].roll_no>p[j+1].roll_no)
        {
           temp=p[j];
           p[j]=p[j+1];
           p[j+1]=temp;
        }
}
void sort2(struct student p[],int n)
{
        int i,j;
        struct student temp;
        for(i=n-1;i>0;i- -)
        for(j=0;j<i;j++)
        if(strcmpi(p[j].name,p[j+1].name)>0)
        {
           temp=p[j];
           p[j]=p[j+1];
           p[j+1]=temp;
        }
}
```

```
void sort3(struct student p[],int n)
{
     int i,j;
     struct student temp;
     for(i=n-1;i>0;i—)
     for(j=0;j<i;j++)
     if(p[j].total<p[j+1].total)
     {
          temp=p[j];
          p[j]=p[j+1];
          p[j+1]=temp;
     }
}
void sort4(struct student p[],int n)
{
     int i,j;
     struct student temp;
     for(i=n-1;i>0;i- -)
     for(j=0;j<i;j++)
     if(p[j].dob.yy>p[j+1].dob.yy||
      p[j].dob.yy==p[j+1].dob.yy&&p[j].dob.mm>p[j+1].dob.mm||
    p[j].dob.yy==p[j+1].dob.yy&&p[j].dob.mm=p[j+1].dob.mm&&p[j].dob.dd>p[j+1].dob.dd)
     {
          temp=p[j];
          p[j]=p[j+1];
          p[j+1]=temp;
     }
}
```


H.W Don't refer Ans

CWP-161 Define a structure called as player that will describe the following player's name, country name, number of matches played and the batting average. Develop a program that will store information of 50 player around the world using this structure. Display names of those player in descending order with respect to batting average.

Ans:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct player
{
     char name[40];
     char country[20];
     int nom;
     float bat_avg;
};
```

```
int main()
{
      struct player p[50];
      struct player temp;
      int i,j,n;
      clrscr();
      printf("How many players(not more than 50)=");
      scanf("%d",&n);
      printf("Enter information of all\n");
      for(i=0;i<n;i++)
      {
            fflush(stdin);
            printf("Name=");
            gets(p[i].name);
            fflush(stdin);
            printf("Country=");
            gets(p[i].country);
            printf("Number of matches=");
            scanf("%d",&p[i].nom);
            printf("Batting average=");
            scanf("%d",&p[i].bat_avg);
      }

      for(i=n-1;i>0;i- -)
      for(j=0;j<i;j++)
      if(p[j].bat_avg<p[j+1].bat_avg)
      {
            temp=p[j];
            p[j]=p[j+1];
            p[j+1]=temp;
      }

      printf("Name                              Bat_avg\n");
      for(i=0;i<n;i++)
      {
            Printf("%s", p[i].name);
            for(j=1;j<=40-strlen(p[i].name);j++)
                  printf(" ");
            printf("%7.2f\n",p[i].bat_avg);
      }
      getch();
      return 0;
}
```

CWP-162 Write a program that uses a structure called point to model a point. Define three point and have the user input values from to two of them. Set the third point equal to the sum of the other two and display the values of the third point. The program should also print the distance between the two points entered by user.

Ans: Refer your notebook for the program

CWP-163 Write a program that uses a structure called point to model a point. Define array of not more than 50 points. Your program should read all these points and should display co_ordinates of those two points between which the distance is minimum

Ans: Refer your notebook for the program

8.4.4 Structures and Pointers –

Using pointers, the structure variables can be accessed indirectly. Consider the following example.

Example–

```
struct student
{
        int roll_no;
        float percent;
        char name[40];
};
struct student s1;
struct student *p;
.....
.....
p=& s1;
.....
.....
```

- In the above example, we have defined **s1** as variable of type **struct student** and **p** is defined of type **pointer to struct student.**
- Consider the following statement,

  p=& s1;

  Due to this statement, the p stores address of s1. In other words, p will act as pointer to s1
- If we want to assign the value 10 to roll_no of s1 then it can be done with three different methods.

  (a) s1.roll_no=10;

  (b) (*p).roll_no=10;

  (c) P→roll_no=10;
- The first method is without using pointer whereas in next two methods we use pointer.
- The meaning of the second and third method, is nothing but roll_no of that structure variable, which is being pointed p. Since **p** points to s1, effectively the second and the third method results into **s1.roll_no.**

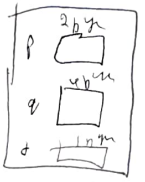
CWP-164: Write a function that accepts pointer to struct student and calculates average of PCM for that student. Write suitable main( ) to read roll_no and marks of PCM for not more than 60 student. The program should calculate average of PCM for every student using a function and should print roll_no, average of PCM and class for every student.

Ans: Refer your notebook for the program.

8.4.5 Unions –

- Union data type is similar to structure data type in many respects. However, there is difference between the two in terms of memory requirements.
- Consider the following example

(a)
```
struct xyz
{
        int p;
        float q;
        char r;
}
struct xyz d;
```



(b)
```
union xyz
{
        int p
        float q;
        char r;
}
union xyz d;
```

- The **example-a** defines a **structure data type xyz** with three members **p q** and **r** of type **int**, **float** and **char** respectively.
- The **d** is a variable of type **struct xyz**. The memory allocated for this variable is the total space taken by all the three member together Hence, the variable **d** will take the storage space of 7 bytes (2 + 4 + 1).
- The values of d.p, d.q and d.r are separately maintained. Hence, the change in value of one of them will not affect the values of other members
- The **example-b** defines a **union data type xyz** with three members **p, q** and **r** of type **int**, **float** and **char** respectively.
- The **d** is a variable of type **union xyz**. It will take space which will be the highest storage space taken by any individual member of union. Since the float member takes the highest space i.e. 4 bytes, the union variable d will take a space of 4 bytes. This space will be shared by **d.p**, **d.q** and **d.r**.
- Hence, when we assign value to any member of **union variable d,** then values of other members are lost.

8.4.6 Differentiate between Structure and Union –

Structure	Union
1. Example - **struct xyz** **{** **int p;** **float q;** **char r;** **};** **struct xyz d;**	1. Example - **union xyz** **{** **int p;** **float q;** **char r;** **};** **union xyz d;**
2. All the members of the structure get individual space.	2. The members of the union will share the space.
3. Memory allocated for the variable d is of 7 bytes which is the addition of the individual space taken by the members.	3. Memory allocated for the union variable d is of 4 bytes which is the highest space taken by the individual member of the union xyz.
4. Structure are advantages when the values of the members are to be maintained simultaneously.	4. Unions are advantageous when the values of all the members are not to be maintained simultaneously. The advantage is that union will take less space.
5. When all the values are not to be maintained simultaneously, then structures are disadvantageous as they require more space	5. When all the values are to be maintained simultaneously, then unions are disadvantageous. This is because union members do not have separate memory to maintain them.
6. Diagram –	6. Diagram -

8.5 Files –

8.5.1 Introduction to files –

- Many applications require data to be written to or read from the secondary storage device (e.g. hard disk). Such information is stored on the secondary storage using data files. Thus, files are useful for permanent data storage.
- The two types of data files are,
  a) stream oriented (or standard) files
  b) system oriented (or binary) files
- The stream oriented files are further classified into two types.
  a) data files made up of consecutive characters
  b) data files made up of blocks containing contiguous bytes of information

8.5.2 Opening and Closing of files –

- While working with files, the first step is to establish buffer area, where information is temporarily stored while transferring data between computer's primary memory and file on secondary storage device.
- The buffer area is established by writing,

  FILE *fp;

  The FILE type will establish a buffer area and the fp will act as pointer to the beginning of that buffer area. The fp will then be referred as file pointer.
- The file must be opened before it is used. This is done with a function called as fopen( ). The typical use of this function can be shown as follows.

  FILE *fp;

  fp = fopen (file name, file type);

  The filename is a string that represent name of the file and the file type is a string that describe the manner in which the file will be opened. The following table describes the manner in which the file can be opened.

File Type	Meaning
"r"	Open an existing file or reading only.
"w"	Open a new file for writing only. If a file, specified with **file name** already exist then it will be destroyed and a new file with same name is created.
"a"	Open a new file for appending (adding information at the end of the file). A new file will be created if the file with the specified **file name** does not exist.
"r+"	Open an existing file for both reading and writing.
"w+"	Open a new file for both reading and writing. If a file, specified with **file name** already exist then it will be destroyed and a new file with same name is created.
"a+"	Open an existing file for both reading and appending. A new file will be created if the file with the specific **file name** does not exist.

Note – If the file is not opened due to any reason then the fopen( ) returns a value 0.

CWP-165: Write a program that will read a line of text and will write its uppercase version into a data file.

Ans: Refer your notebook for the program.

CWP-166: Write a program that will read character data from a data file and will print those characters on the output device. The program should also display how many of the characters are vowels, consonants, digits and the other characters.

Ans: Refer your notebook for the program.

CWP-167: Write a program that will read integer numbers from a data file and will print highest and lowest out of it on the output device.

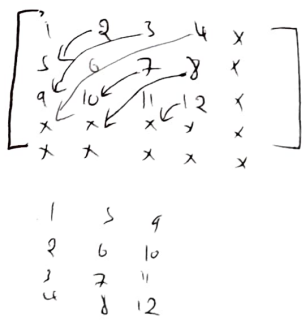
Ans: Refer your notebook for the program.

CWP-168: Write a program that will read name and marks of physics, chemistry and maths separately stored in a file and will store the data along with the total of PCM in the other file.

Ans: Refer your notebook for program.

CWP-169: Write a program that opens a file for appending. The program should read character type data which will be terminated with character #.

Ans: Refer your notebook for the program.



Extra

WAP to copy the data from one file to other

WAP to accept a name and roll no of a student and store in a text file and read the stored data and display the same from the file