Introduction.

1. Commonly operating machine particularly used for technical educational and research.
2. The computer is a word came from a latin word compute which is firm calculator
3. In simple word, a device which performs the calculations. C.P.U is nothing but micro processor, it is the heart of the computer. Actually it does the work in the processing unit which is arranged inside of the processing unit of on the mother board
4. The knowledge of applications of the computers are known as software technology
5. System software is nothing but the programs which are support to run the system, in simple words.
6. Operating system is a set of program's which acts as an interface between user and hardware as well as which provides an environment for executing the user commands.
7. Operating system is acting as a manager for both software and hardware resources
8. Application softwares are nothing but the prograny which are support to develop the applications usually languages. Package is a pre defined program designed by somebody else and used by us.
9. Languages are used to communicate with the system in order to construct the programming structure

- Computer Technology
  - Software Technology
    - System software
    - Application software
      - Packages
      - languages
  - Hardware technology

Scanned by CamScanner

languages:

software <--- programs <-- instructions <--- languages

Eg; C, C++, VC++, C#, Java, Net, Android.

Q) Why C is ?

→ C language is developed by Mr. Dennis Richet AT and T Bell ba labs USA in 1970's.

→ With the help of 'c' language, we can develop business, commercial, scientific and engineering purpose programs.

→ Does not required to write lengthy programs for small calculations

→ C-programs are executing with high efficiency.

→ C-programs are portable or transportable

→ C is architecture independent and also C is operating system dependent.

Types of languages:

Languages

Machine level language	Assembly language	High level language
0's and 1's only binary digits low level language 1000001	Digits + symbolic code 0 to 9 and ADD, SUM, MUL, DIV	A TO Z, a to z, 0 to 9, >, <, >=, <=, ==, ... Eg; C, C++, JAVA

→ Machine languages:

(1) Languages which are understandable by machine.

(2) It can understand only 0's and 1's

(3) These are also known as binary digits or low level languages

(4) Machine languages are difficult in learning, implementation and modifications.

(5) In order to work machine language we must require the knowledge of system architecture

Scanned by CamScanner

(6) These are specially designed for circuit programs
(7) The programs which are developed in machine languages with will be executed very fast.

Assembly languages:

(1) It is combination of both digits and pneumonic codes.
(2) These are specially designed for developing the programs related to microprocessor.

High level languages:

(1) It will support alphabets, digits, symbols, special characters
(2) High level language instructions are regular english instruction
(3) Inorder to work with the high level languages, doesnot require to have knowledge of any system architecture.
(4) These are easy to learn, implement and make the modification
(5) These are specially meant for developing application softwares.

Q) What is a translator?

Translators are used to translate source code to object code where source code is nothing but the code which is written by us in high level language, object code is nothing but the code which is machine understandable

Translators are classified into 2 types:

(1) Interpretor (2) Compiler.

→ Interpretor are used to translate source code to object code consequently if at all any errors occur then the program execution will stop there till it is error free.

→ Compilers are used to translate entire source code to object code in a single attempt. If at all any errors are occur they are isolated on the next screen.

→ The compilers are faster than interpreters.

NOTE: C is a high level language and it is a compiler

Scanned by CamScanner

C TOKENS:

→ A token is nothing but smallest individual component in the program

→ The C languages offers the following tokens

(1) Character set
(2) Variables
(3) Data types
(4) Operators
(5) Identifers
(6) Key words

CHARACTER SET:

It represents set of characters which are supported by the corresponding programming language.

C language will support ASCII character set

ASCII - American standard code for information interchange

Range: 0 to 255 which includes

A TO Z : 65 to 90
a to z : 97 to 122
0 to 9 : 48 to 57
space bar : 32
Escape : 27
backspace : 8
Enter : 13

VARIABLE:

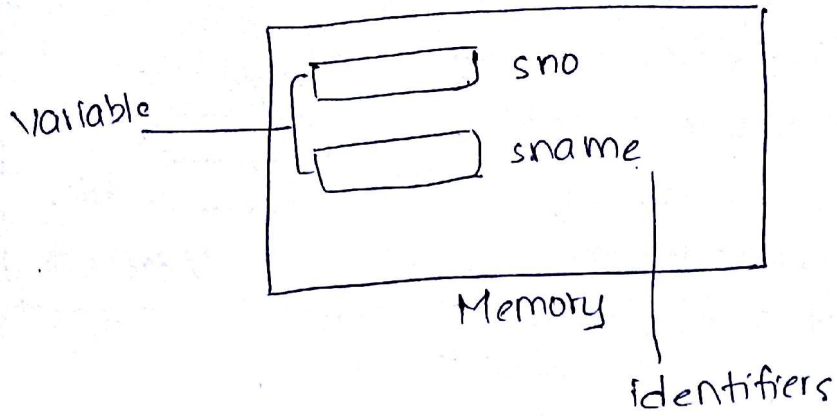
It is a named container which enables you to storage the data temparorily douring the program execution.

In simple words a variable is nothing but space to store the data

Scanned by CamScanner

ANSI - American National standard Institution

syn: [storage class] b̸ <datatype> b̸ <vname> [=value];



Data types:

It represents type of data to be allowed in the specified memory location. In general data types are classed into 3 types

(1) Pre defined datatypes
(2) Abstract datatypes
(3) User defined datatypes

(1) Predefined datatypes: provided by the turbo C2 incorporation along with the C compiler

Predefined datatypes are classified into 3 types

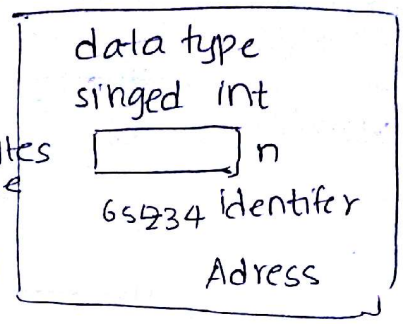
→ Integer types:

Integer types are recommended to use whenever you want to store a number without decimal. These are classified into 4 types based on their sizes and ranges

a) signed int:
- It will take 2 bytes of space in memory
- It will allow both positive and negative numbers whose range is from -32768 to 32767
- format specifier is %d
- eg: -25, 0, 89, 2345

Scanned by CamScanner

Syn : <data type> <vname> =[value];
Eg : singed int n; or
int n;



NOTE:

Whenever we declare a variable then compiler automatically initialise the value for the declared variable called garbage value.

(b) unsigned int :
- It wike take 2 bytes of space in memory
- whose range is from 0 to 65535
- It will allow only positive integers
- format specifier is %u

Eg : 254, 32765u.

(c) long integer :
- It will take 4 bytes of space in memory
- It will allow both positive and negative numbers
- Its range is from -2,147,483,648 to 2,147,483,467
- format specifier is %ld

Eg: 45l, -5l, 40000l

(d) unsigned long :
- It will take 4 bytes of space in memory
- It will allow only positive integers
- Its range is from 0 to 4,294,967,295
- format specifier is %lu
- Eg: 100lu, 51lu.

→ Real Types :

Real types are used to store a number with decimal. These are mostly recommended to use scientific values of exponential values. These are classified into 3 types

Scanned by CamScanner

a) float:

→ It will take 4 bytes of space in memory

→ Whose range is from $\pm 3.4 \times 10^{\pm 38}$

- It will allow 6 decimal places after the dot.
- format specifier is %f
- Eg: -3.5f, 7.5f

b) double:

- It will take 8 bytes of space in memory
- whose range is from $\pm 1.7 \times 10^{\pm 308}$
- It will allow 15 decimal places after the dot
- format specifier is %lf
- Eg: -3.5, 7.5

c) long double:

- It will take 10 bytes of space in the memory.
- whose range is from $\pm 3.4 \times 10^{\pm 4932}$
- It will allow 19 decimal places after the dot.
- format specifier is %Lf
- Eg: -3.5l, 7.5l

→ character types:

character types allows you to store α numerical values.

Character types are classified into 2 types based on the range

(a) singhed type character

(b) unsigned type character

- The default type of character is signed.
- Both will take 1 byte of space in the memory.
- Format specifiers for both signed and unsigned character is %c
- Range for signed char is -128 to 127. For unsigned character it is 0 to 255.

NOTE:

If we enter any value in ther character type then they are treated by the system as a character only, hence

Scanned by CamScanner

we require some typecasting techniques in order to maniplate the data existed in the character variable

→ Identifiers:

All the names which are declared by us for our programming requirements such as variable names, structure names, file names etc.

limitations:

1) It must began with an alphabet or an underscrole.
2) Its length donot cross 32 characters.
3) It may be in uppercase or lowercase letters or mixed case.
4) It may have a special character that is an underscrole
5) It may have digits
6) It should not be the clang related word (Key words)

→ Key words:

- Keywords are nothing but reserved words.
- The meaning of keywords already defined by the compiler.
- Every keyword is having its unique importance in the program.
- The meaning of the keywords cannot be modified

asm, break, case, cdecl, char, const, continue, default, do, double, else, enum, extern, far, float, for, goto, huge, if, int, interrupt, long, near, pascal, register, return, short, signed, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

Scanned by CamScanner

OPERATORS:

→ Operator is a symbol used to perform the operation on the operands

→ Basically operators are classified into 2 types.

(1) Uniary operators

(2) Binary operators

→ An operator which is used to perform operation on a single operand is known as uniary operator where Binary operator is an operator used to perform operation on more than one operand minimum of two.

→ Types of operators:

(1) Arithematic operators

(2) Relational operators

(3) Logical operators

(4) Assignment operator

(5) Conditional operator

(6) Increment / Decrement operator.

(7) size of operator

(8) compound operators

Assignment operator:

It is used to assign a value or a expression

Assignment operator is =

Assignment always takes place from right to left

Eg: int m1; // variable declaration

m1 = 10; // Assign value

int m1=10; variable declaration cum intialization.

int x,y,z;

x=10;
y=10;
z=10;

x=y=z=10;
↳ multi Assignment

10	10	10
x	y	z

Scanned by CamScanner

Arithematic operators:

These are used to perform all arithematic expressions

int a=10, b=5;

operator	Action	Result
+	a+b	10
-	a-b	5
*	a*b	20
/	a/b	2
%	a%b	0

int/int=int

int/float=float

float/int=float

float/float=float

int a=5; float b;

b=a/2;

output = 2.000000

b=a/2.0f;

output: 2.500000

Relational operators:

These are used to check the relation among the operands and it will return bolean values (0-flase, 1-true)

int x=10, y=20;

operator	action	result
>	x>y	0
<	x<y	1
>=	x>=y	0
<=	x<=y	1
==	x==y	0
!=	x!=y	1

Eg:

int x=10, y=20, z;

z=x<y;

z=1

int a; a=10<20!=10;

a=1

Eg: int d;

d=10!=10==10<=(2*2)+1;

d=1

Scanned by CamScanner

Logical Operators:

These are used to validate more than one condition. The C-language will offer the following logical operators.

1) NAND (&&)
2) OR (||)
3) NOT (!)

→ AND is recommended to use when all the list of conditions required to evaluate to true.

→ Atleast one condition in the list need to be satisfied. In all such cases we need to make use of an operator "OR".

→ NOT will work quiet opposite for regular relational operator result.

int x=10, Y=20, Z=30;

AND (&&)

EXP 1	EXP 2	RES
(X>Y) 0	(X>Z) 0	(X>Y && X>Z) 0
(Y>X) 1	(Y>Z) 0	(Y>X && Y>Z) 0
(Z>X) 1	(Z>Y) 1	(Z>X && Z>Y) 1

(OR) (||)

EXP 1	EXP 2	RES
(X>Y) 0	(X>Z) 0	(X>Y || X>Z) 0
(Y>X) 1	(Y>Z) 0	(Y>X || Y>Z) 1
(Z>X) 1	(Z>Y) 1	(Z>X || Z>Y) 1

Scanned by CamScanner

NOT(!)

int X=0, Y=20, z;

z=X>Y; 0

z=!(X>Y); 1

z=(10<20);

z=1

z=!(10<20)

z=0

int X;

X=(10<10|| 5>=10);

0||0

X=((2+3)!=(3+2)&&10<=10);

5!=5 && 10<=10

0 && 1

X=(2&&3) — [other than 0 it is true
2=true
3=true]

=1

Compound Operators:

→ These operators works exactly similar with arithematic operators.

→ These are the combination of both arithematic and assignment operators.

→ These are also known as shortcut operators.

Operator	Action	Result
+=	X+=Y	X=X+Y
-=	X-=Y	X=X-Y
*=	X*=Y	X=X*Y
/=	X/=Y	X=X/Y
%=	X%=Y	X=X%Y

Scanned by CamScanner

→ Conditional operators:

- It will check the condition initially, if the condition evaluates to true it will execute logical true otherwise it will execute logical false.
- Another name for the conditional operator is ternary operator
- The operator is ? :
- (<condition>) ? logical_true : logical_false;

  true → logical_true, false → logical_false

- int x=100, y=20;
  (x>y) ? Printf ("x is big") : printf ("y is big");
- int marks=98;
  (marks > 34) ? printf ("Pass") : Printf ("Fail");
  (marks < 35) ? print ("Fail) : Printf (" Pass);
- int a=2, b=3, c;
  (a<b) ? c=a+b : c=a*b; // invalid
  (a<b) ? (c=a+b) : (c=a*b);
  c = (a<b) ? c=a+b : c=a*b;

→ Increment and decrement operator:

Increment operator is used to increase a value by a value.
The Increment operator is ++

Eg: int x=10;
x = x+1; or
x++;

Decrement operator is used for decrease by a value.
The decrement operator is --

Eg: int x=10;
x = x-1; or
x--;

Scanned by CamScanner

Increment and decrement operators are classified into 2 types:

Increment

- Preincrement: ++X
- Post increment: X++

Decrement

- Predecrement: --X
- Post decrement: X--

In pre increment, pre decrement the value of ~~value~~ of variable will be ~~d~~ increased or decreased by 1 before initialization or assigning the value.

In post increment, post decrement the value of the variable will be increased or decreased by 1 after assiging or intializing the value.

	X (10)	Y (20)	Z (30)
Y = ++X;	11	11	30
Z = --Y;	11	10	10
X = ++Z;	11	11	11
Y = X++;	12	11	11
Z = Y--;	12	10	11
Y = Z++;	12	11	12

Pre increment and post increment remains same, lly pre decrement and post decrement remains same when that expression is not having the assignment operato

	X	Y
	10	
Y=X++;	11	10

	X
	10
X++	11

	X	Y
	10	
Y=++X	11	11

	X
	10
++X	11

Scanned by CamScanner

sizeof operator:

WKI it will return number of bytes are taken by a variable or a data type and also it will return number of bytes taken by a function

RULES FOR WRITING THE C PROGRAMS:

1) Usually all the programming statements must be written in lowercase letters
2) Every program execution must began with main() function
3) All the programming statements must be written in between open and closed curly braces ({ })
4) List of variables of same type should be seperated by a comma (,)
5) Every statement should end with semi colon.(;)
6) Every function name should end with couple of paranthesis (())
   Ex: clrscr(); print(); scanf();
7) while comparing or assigneding a character should be taken in signgle quotation marks 's'
8) While comparing or assigning a string should be taken in double quotation marks "sssIT."
9) Comments should be taken in between /* comments* /

Scanned by CamScanner

Structure of c-program:

```
/* comment section */
File inclusions
Pre processor directives
global declaration
void main()
{
[local variables];
statements are group of statements (s);
  |
}
[return type] <function name> ([list of args])
{
[local variables];
statement(s);
[return <value/variable/exp>];
}.
```

→ comment section:

- In the comment section we can write anything related to the program.
- We can write the comment section any where in the program and 'n' number of times in the program.
- The statements which are taken in between /* */ those statements are not participated in the program execution.

NOTE: Writing a comment inside a comment is not supported

Scanned by CamScanner

→ File inclusion:

1) In c-lang we have collection of libraries.
2) A library is nothing but collection of functions.
3) Every function is a self contained block or a sub program.
4) whenever you want to work with any function related to a particular library, then it is recommended to include the corresponding headerfile as a file inclusion
5) If you fail to include the corresponding header file in c++ compiler will raise a warning i.e,
THE FUNCTION SHOULD HAVE A PROTOTYPE

" c-language"

stdio.h	conio.h	string.h
printf();	clscr();	
scanf();	getch();	

```
# include <stdio.h>
main()
{
Printf ("Hello");
Printf ("Bolo");
}
```

Input and output functions:

Input functions:

Input functions are used to read the values from the console (device) and stores them into the variable

Ex: scanf (f-format)

Scanned by CamScanner

Output functions:

Output functions are used to display the messages on console (device)

These are used to read the values from the variables and displayed on the console

Ex: Printf() (f-format)

NOTE: Printf function will return integer type bytes to o/s. The scanf function always used to return no of inputs are taken from user

SYNTAX:

scanf(" control strings", &variables);

printf(" [msg/control strings]", [variables]);

→ Format specifiers:

- These are also known as control strings
- control strings are used to control various type of values during input and output.
- Implicit conversion will be done by the compiler integer to characters and character to integers

signed int	%d	float	%f
unsigned int	%u	double	%lf
long int	%ld	long double	%Lf
unsigned long	%lu		

char

signed char	%c
unsigned char	%c
string	%s

int x = 100; $\boxed{100}$ x

printf("x"); --> x

printf(x); --> error

printf("2+3"); --> 2+3

printf("%d", 2+3); --> 5

printf("50*2"); --> 50*2

printf("%d", 50*2); --> 100

Scanned by CamScanner

```
printf("%d",x); --> 100
printf("%d", 100); --> 100
printf("100"); --> 100
```

```
printf("x value is");
--> x value is
printf("x value is %d",x);
--> x value is 100
```

```
int x=65;
printf("%d",x); 65
printf("%c",x); A
```

```
printf("%c", 32+33); A
printf("%d", 'A'+'B'); 131
```

```
char x='A';
printf("x"); x
printf("%c", x); A
printf("%d",x); 65
```

```
int a=5; float b;
b=a/2.0f; or
b= (float) a/2;
```

→ Eg:

```
int sno;
char sname[20];
```

100	sno
sai	sname

```
scanf("%d", sno);
scanf("%s", &sname);
scanf("%d %s",&sno, &sname);
printf("%d", sno);
printf("%s", sname);
printf("%d %s", sno,sname);
printf("%s %d", sname, sno);
printf("%d %s", Sno, SNAME); //Error.
```

Scanned by CamScanner

Installation:

step1: copy TURBO C2 and DOSBOX from CD to local disc (c:)

step2: Install DOSBOX

step3: Open ~~BO~~ DOSBOX

step4: Z:\>~~mono~~ mount c c:\ <--enter

Z:\c:\ <---enter

C:\>CD TURBOC2 <--enter

c:\TURBOC2>TC <--enter

Program 1:

1. →

```
/* To Print A Message */
#include<stdio.h>
main()
{
Printf("Welcome");
}
```

Alt + C - compilation

Alt + R - run and execute

Alt + F5 - output

Alt + F2 - save

→ Backslash characters:

These are used to produce a result in the user desired fashion.

These are strongly recommended to use in output functions.

These are also known as escaped sequences.

It should not be used in input functions. If we use them, it won't give any compile time error.

Back slash char	Respone
\n	Newline
\t	Horiz tab(8)
\a	bell sound
\'	It will print ' sym
\"	It will print " sym
\\	It will print \ sym
\?	It will print ? sym

Scanned by CamScanner

Program 2:

```
/* To Print Message */
# include <stdio.h>
main()
{
printf ("welcome \"TO"\ sssIT \n");
printf ("GOOD DAY");
printf ("\n Nice Day  to\n\n\n It \t  \"My Dear\" ");
}
```

Output:

Welcome "TO" SSST
GOOD DAY
NICE DAY
.
.
. My Dear'

- .obj file is created after compilation.
- .EXE file is created after execution

Program 3:

```
# include <stdio.h>
main()
{
printf ("\\\'\'\\\?\\");
}
```

compile time error - unterminated string

Program 4:

```
# include <stdio.h>
main();
{
printf ("\\\'\'\\\"");
printf ("\n\?\?\\");
}
```

error because of ; after main()

Scanned by CamScanner

Programs:

```
#include <stdio.h>
main()
{
printf ("\n AB");
printf ("\bSI");
printf ("\r HA");
}
```

carraige return – \r
↳ brings the curser to starting point

Output: HAI

AB
ASI
HAI

→ getch();:
It is an input function waiting for getting the character from the console without echoing

→ getche();:
It is an input function waiting for getting the character from the console with echoing.

NOTE:
This two functions are supported by conio.h prototype

→ clrscr();:
It is used to clear text mode window or to clear the screen. It is also from conio.h.

NOTE:
As per the programing convenience we can use a function 'n' number of times anywhere in the program but there is a limitation for clrscr function in c-compiler. It is recommended to write after variable declaration.

Scanned by CamScanner

Program 6:

```
#include<stdio.h>
#include<conio.h>
main()
{
 clrscr();
 printf("\n welcome");
 getche();
 printf("\n to");
 getche();
 printf("\n SSSIT");
 getch();
}
```

output:

```
welcome a
 to m
 SSST
```

→ Interacting with variables:

Program 7:

Accept a program user character from user and print it on screen.

```
#include <stdio.h>
#include <conio.h>
main()
{char n;
 clrscr();
 printf("\n enter any character");
 scanf(" %c", &n);
 printf("\n given character is: %c", n);
 getch();
}
```

Scanned by CamScanner

Program 8:

Accept a string from the user and print it on the screen

```
#include <stdio.h>
#include <conio.h>
main ( )
{
char n[20];
clrscr();
  printf("\nenter the string");
  scanf(" %s",&n);
  printf("\nthe given string is:%s", &n);
getch();
}
```

NOTE:-

writing address operator is optional in the scanf while extracting string type of values from the user.

Programa:

Accept a number from the user and print it on the screen

```
#include <stdio.h>
#include <conio.h>
main( )
{
int n;
clrscr;
  printf("\n enter a number");
  scanf(" %d", &n);
  printf("\n given number is: ", &n);
getch();
}
```

If 332768 is given then the output is -32768

32769 -32767

Scanned by CamScanner

Program 10:

Accept a number from the user and find the square of a number.

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,s;
 clrscr;
 printf("\n enter a number");
 scanf(" %d", &n);
   s=n*n;
 printf(" \n Result is : %d", s);
 getch();
 }
```

Program 11:

Accept the radius of a circle and find out area of circle.

```
#include <stdio.h>
# include <conio.h>
main()
{
float rad, area;
 clrscr;
 printf ("\n enter the radius");
  scanf (" %f", &rad);
  area = (3.14 * rad * rad)*
   Print f ("\n area of circle is %.3f ",area);
  getch();
 }
```

Scanned by CamScanner

Program 12:

Accept the radius of circle and find the circumference of circle

Program 13:

Accept 2 numbers from the user, find $(a+b)^2$

```
main()
{
int a,b,r;
clrscr();
printf("enter two numbers");
scanf(" %d %d ", &a,&b);
r= (a*a)+(b*b)+(2*a*b);
      (or)
r= (a+b)*(a+b);
printf(" result is %d", r);
getch();
}
```

Program 14:

Accept 3 subject marks from the user, calculate total and average.

```
main()
{
int m1,m2,m3, tot;
float avg;
clrscr();
printf("enter three subject marks");
scanf ("%d %d %d", &m1, &m2, &m3);
tot = m1+m2+m3;
avg= (tot)/3.0f;
printf(" \n total marks is: %d", tot);
printf(" \n avg is:%.2f", avg);
getch();
}
```

Scanned by CamScanner

Programis:

Accept present reading, previous reading. calculate number of units and find bill amount. Unit - Rs 2/-

```
main()
{
int prr, pvr, nu, bi;
clrscr();
printf("\n enter present and previous reading");
scanf("%d%d", &prr, &pvr);
   nu= prr- pvr;
   bi= nu*2;
printf("\n number of units: %d", nu);
printf("\n bill amount: %d", bi);
getch();
}
```

Programis:

Accept number of pens are purchased, number of books are purchased. calculate bill, discount and ~~then~~ net bill

pen cost = 25 book cost = 50 , discount - 10%

```
main()
{
int npp, nbp, bi, amt, change;
float dis, net;
clrscr();
printf("\n enter npp and nbp"),
scanf("%d,%d", &npp, &nbp);
 bi= (npp*25) + (nbp*50);
 dis= (bi*10)/100;
 net= bi-dis;
 clrscr();
```

Scanned by CamScanner

```
printf("\n = = = = = = = =");
printf("\n AMBICA SUPER MARKET");
printf("\n KPHB-HYD");
printf("\n = = = = = = = =");
printf("\n No.of pens : %d", npp);
printf("\n No.of books : %d", nbp);
printf("\n - - - - - - - - - - - - - -");
printf("\n Bill amount is : %d", bi);
printf("\n Discount is : %.2f", dis);
printf("\n Net Bill is : %.2f", net);
printf("\n - - - - - - - - - - - - - -");
printf("\n Enter amount from client");
scanf("%d", &amt);
change= amt-net;
printf("\n change : %d", change);
getch();
}
```

Output:

```
= = = = = =
AMBICA SUPER MARKET
KPHB-HYD
= = = = = =
No.of pens : 2
No: of books : 3
Bill amount : 200
Discount : 20
Net Bill = 180
- - - - - - - - - - - -
Enter amount from client. 500
change : 320
```

Scanned by CamScanner

Program 17:

```
main()
{
int main=100;
clrscr();
printf("\n%d", main);
getch();
}
```

output : 100

main is not a keyword.

Program 18:

```
main()
{
clrscr();
pf("\n2+3");
pf("\n%d", 2+3);
pf("\n2+3=%d;2+3);
getch();
}
```

output

2+3

5

2+3=5

Program 19:

```
main()
{
int a=10, b=20;
clrscr();
printf("\n%d %d %d", a, b);
getch();
}
```

10, 20, (some garbage value will be printed)

~~10, 20, 26~~

Scanned by CamScanner

Program 20:

```
main()
{
int a=10,b=20;
clrscr();
printf("\n %d %d");
getch();
}
```

output: 10,20 → only in TURBO C2

Program 21:

```
main()
{
clrscr()
printf("\n %d %d");
getch();
}
```

output: -28 285 (grabage values)

program 22:

```
main()
{
int a=10,b=20;
clrscr();
printf("\n %d %d");
getch();
}
```

out: 10,20, 285 ↳ garbage value

program 23:

```
main()
{
clrscr();
printf("%d", printf("welcome"));
getch();
}
```

output:
welcome 7
↳ garbage value

Scanned by CamScanner

Program 24:

```
main()
{
clrscr();
printf("%d", printf(" 10 20"));
getch();
}
```

output 10 20 5

Program 25:

```
main()
{
int a,b,c,d;
clrscr();
d=scanf("%d %d %d", &a,&b,&c);
printf("\n %d %d %d %d", a,b,c,d);
getch();
}
```

output 10 20 30 3

program 26:

```
main()
{
int a,b,c,d;
clrscr();
printf("%d", scanf("%d %d %d",&a,&b,&c));
printf("\n %d %d %d", a,b,c);
getch();
}
```

output:
```
10
20
30
3
10 20 30
```

Scanned by CamScanner

Program 27:

```
main()
{
int a;
clrscr();
a=10,20,30;
printf ("\na=%d",a);
getch();
}
```

output = 10

program28:

```
main()
{
int a;
clrscr();
a = (10,30,20);
printf ("\na=%d",a);
getch();
}
```

output = 20 ↳ last element in the given set of variables

program29:

```
main()
{
int a;
clrscr();
a=300*300/300;
printf ("\n a=%d", a);
getch();
}
```

300 * 300 = 90,000
-35767
-26767
( )/300
= 81.523
= 81

output : 81

Scanned by CamScanner

Program30:

```
main()
{
int a=10;
clrscr();
printf("\n %d %d %d", ++a, ++a, ++a);
printf("\n %d %d %d", ++a, a++, a++);
printf("\n %d %d %d", a++, a++, ++a);
printf("\n a=%d", a);
getch();
}
```

output:

13 12 11

16 14 13

18 17 17

a=19

Program31:

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

output a= 13 b= 39

∵ 3 premcremenents

Program32:

```
main()
{
int a=10, b;
clrscr();
b= ++a + --a + ++a;
printf("\n a=%d b=%d", a b);
getch();
}
```

output: a= 11, b=33

∵ 2 preincrements

1 predecrements

10 → 11 → 12 → 11

Scanned by CamScanner

Program 33:

```
main()
{
int a=10, b;
clrscr();
 b=a-- + ++a + ++a;
 printf ("\n a=%d b=%d ", a, b);
 getch();
}
```

a=11 b=36

preincrement → assig addition → assignment → post increment
decrement
decrement

Scanned by CamScanner

CONDITIONAL STATEMENTS

If you want to perform any calculations or operations based on certain condition then we need to make use of conditional statements.

Conditional statements also known as decision making statements.

C-lang offers the following conditional statements.

(i) if

(ii) if else

(iii) If else if

In general "if" it will check the conditional initially, if the condition evaluates to true it will execute the following block of statements and it will comes out. If at all any statements are given outside of the block then those statements will also participated in program execution.

Syntax:

```
if (<condition>)
{
statement(s);      true
}
false
statemen (s);
```

Program 34:

Accept a number from the user and print it if the given number is positive.

```
main()
{
```

```
int n;
clrscr();
printf("\n enter any number");
scanf("%d", &n);
if (n>0)
{
  printf("\n given no is %d", n);
  }
getch();
}
```

Program 35:

Accept age from the user, print chaild if age is <13 years

```
main()
{
int age;
clrscr();
print("\n enter age");
scanf("%d", age);
  if (n<13)
  printf("\n child")
  getch();
```

Program 36:

Accept salary from the user calculate bonus and net salary. Bunous is 1000 Rs if salary is greater that salary. 10,000

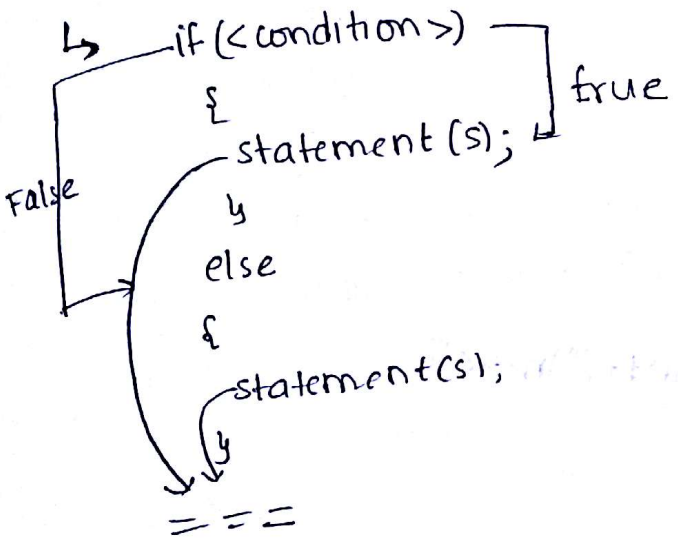
```
main()
{
int  sal, bo=0, net;
clrscr()
print("\n enter the salaray");
```

Scanned by CamScanner

```
scanf ("%d", &sal).

if (-if
  if (sal >10,000)
   {
     bo=1,000;
   }
  net = sal+bo;
  printf (" \n net sal: %d", net);
  getch();
}
```

→ <u>If else</u>

SYNTAX:



As per the if else if will evaluates the condition first, if the condition evaluates to true it will execute the following block of statements after executing those statements and the control will comes out from the scope and the statements which are given outside of the scope will be executed.

if the condition evaluates to false, it will execute the else block. after executing the else block it will come out from the scope.

Scanned by CamScanner

Program37:

Program to find out the given number is +ve or -ve.

```
main()
{
int n;
clrscr();
printf("enter the given number");
scanf("%d",&n);
if (n>0)
{
printf("\n+ve");
}
else
{
printf("\n-ve");
}
```

(or)

```
(n>0)? printf("\n+ve") : printf("\n-ve");

getch();
}
```

Program38:

Accept a char from user, print male if the given char is 'm' else female.

```
main()
{
char g;
clrscr();
printf("enter the any char");
scanf("%c",&g);
```

Scanned by CamScanner

```
if (g=='m');
{
printf("\n MALE");
}
else {
{
printf("\n FEMALE");
(or)
(g=='m') ? printf("\n MALE"): printf("\n FEMALE");
getch();
}
```

Program39:

Accept age from the user ;print major if age is greater than 21 years.

Program40:

Accept marks from the user, print pass if marks are greater than 34. else fail

Program 41:

Accept a number from the user, find out the given number is odd or even

```
main()
{
int n;
clrscr();
printf(" enter the number");
scanf(" %d", &n);
{
for (n%2==0)
{ printf("\n EVEN");}
else
{ Printf("\n ODD");}
getch();
```

(or)

```
(n%2==0)? printf("\n EVEN"): Printf("\n ODD);
```

Scanned by CamScanner

Program 42:

Accept a year from the user, find out the given year is leap year or not.

→ Working with multiple conditions:

Program 43:

Accept a character from the user print male if the given character is m or M else female.

```
main( )
{
char g;
clrscr();
printf (" enter the character ");
scanf (" %c", &g);
if (g=='m' || g=='M')
{ printf ("\n MALE "); }
else
{ printf ("\n FEMALE "); }
```

(or)

```
(g=='M' || g=='m')? printf ("MALE"): printf ("FEMALE");
getch();
}
```

Program 44:

Accept a character from the user, find out the given character is constnant or vowel

```
main( )
{
char 'g';
clrscr();
```

Scanned by CamScanner

```
Printf (" enter the character");
scanf ("%c", &g);
if (g=='a' || g=='A' ||
    g=='e' || g=='E'||
    g=='i' || g=='I'||
    g=='o' || g=='O'||
    g=='u' || g=='U' || )
  { Printf ("\n VOWEL"); }
  else
  { Printf (" \n CONSONANT"); }
  getch();
  }
```

Program 45:

Accept a number from the user find out the given number is divisible by 3 or 7

```
main( )
{
int n;
clrscr();
printf("enter any number");
scanf ("%d", &n);
if (n%3==0 || n%7==0)
{ printf ("\n DIV"); }
else
{ printf ("\n NOT "); }
getch();
}
```

Scanned by CamScanner

Program 46:

Accept 3 subjects marks from the user, find out result. The pass marks >34 in all subjects.

```
main()
{
int m1, m2, m3;
clrscr();
printf("enter the 3 subject marks");
scanf("%d %d %d", &m1, &m2, &m3);
If (m1 > 34 && m2 >34 && m3 >34)
{ printf ("\n PASS"); }
else
{ printf ("\n NOT"); }
getch()
}
```

If --- elseIf:

```
If (<condition>)
{
statement(s);
}
else if (<condition>)
{
statement(s);
}
else
{
statement (s);
}
======
```

true

false

true

false

Scanned by CamScanner

Program 47:

```
main()
{
 int n;
 printf(" enter the number");
 scanf(" %d", &n);
  if (n==1)
   { printf ("\n one"); }
   else if (n==2)
   { printf ("\n two"); }
   else if (n==3)
   { printf ("\n three"); }
   else
   { printf ("\n Invalid"); }
   getch();
   }
```

```
(n==1)? printf("one"):
(n==2)? printf("two"):
(n==3)? printf("three"):
printf("Invalid");
```

Program 48:

Accept age from the user and display the status of the candidate

if age < 5years print kid

if age < 13years print child

if age < 20 years print teenage

if age < 50 years print middle age

else old age

Program 49:

Accept serve number from the user and print an appropriate service name

if service no: is 100 print police

if service no: is 101 print fire

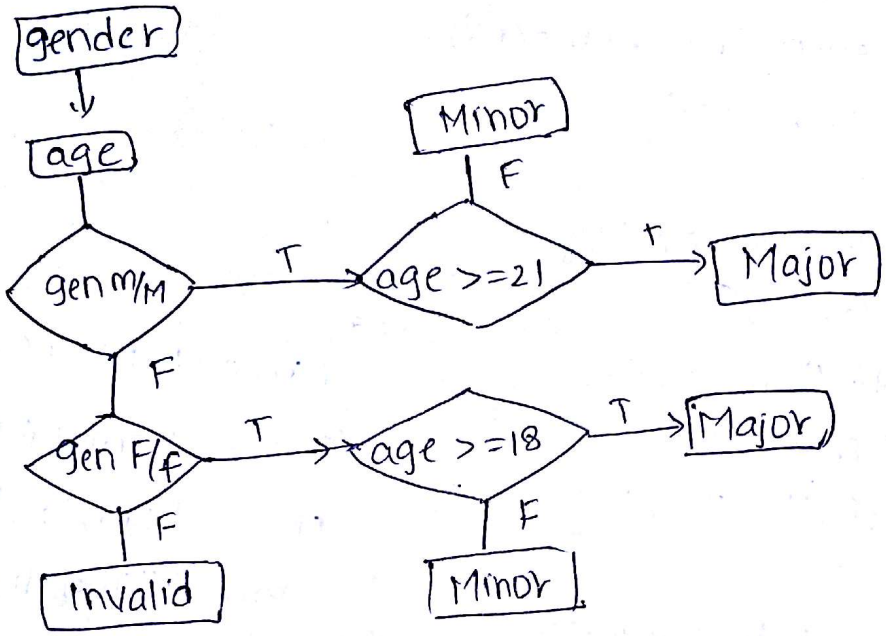
if service no: is 104 print medical

if service no: is 108 print EMRI else Invalid

Scanned by CamScanner

Program 50:

Accept gender from the user, age and find out status of the candidate.



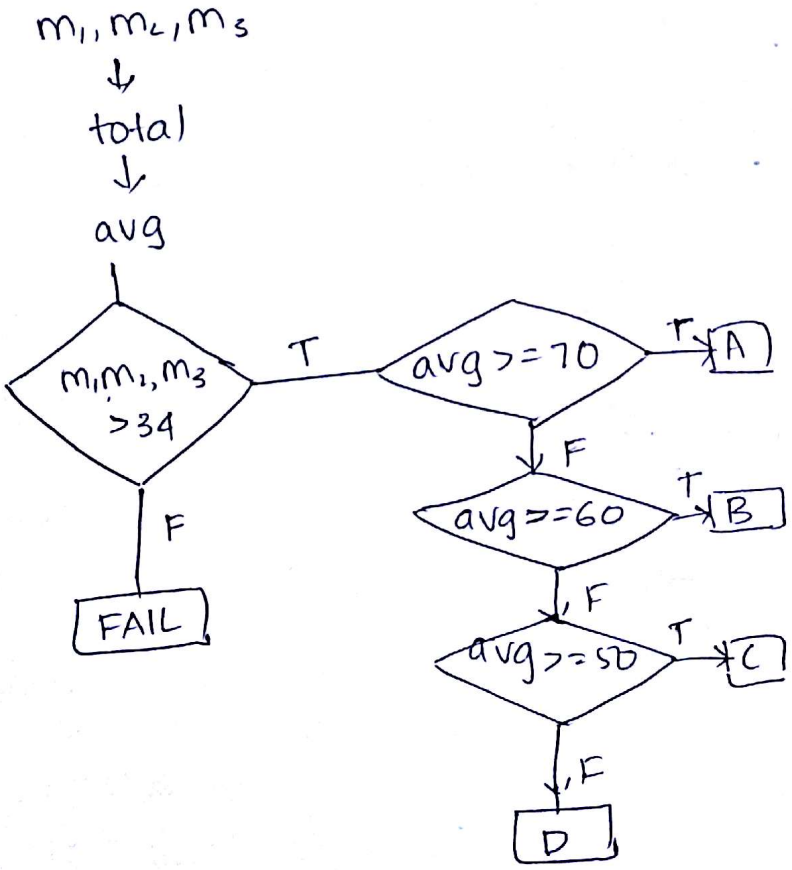

```
main( )
{
char g;
int a;
printf (" enter the gender ");
scanf (" %c", &g);
printf (" \n enter the age");
scanf (" %d", &a);
if (g=='m' || g=='M');
 {
  if (a>=21)
  printf ("\n major");
  else
  printf (" \n minor");
 }
```

Scanned by CamScanner

```
else if (g==='F' || g==='f')
{
if (a>=18)
printf("\n major"),
}
else
{ printf("\n Invalid"); }
getch();
}
```

Program 51:

Accept 3 subject marks from the user. calculate total, average and final out grade



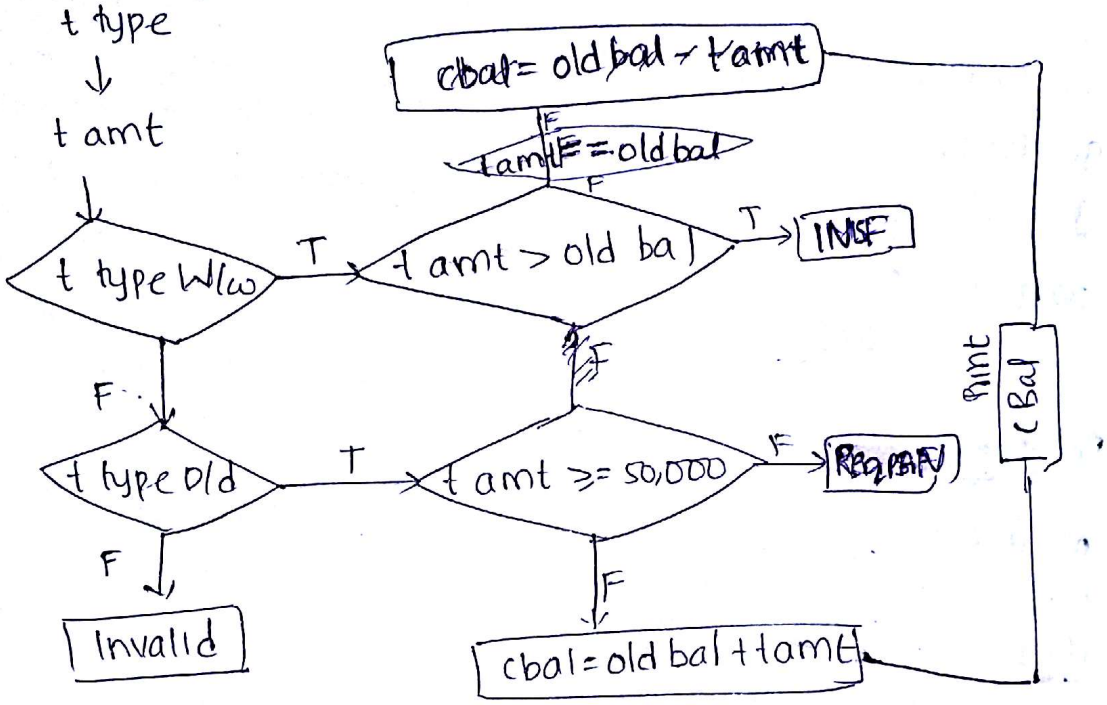
Scanned by CamScanner

```
main()
{
int m1,m2,m3,tot;
float avg;
clrscr();
printf("enter m1,m2,m3");
scanf("%d %d %d", &m1, &m2, &m3);
    total= m1+m2+m3;
     Avg= tot/3;
if (m1>34 && m2>34 && m3>34)
 {
 if (avg>=70)
  printf("\n GRADE:A");
  else if (Avg>=60)
   printf("\n GRADE:B");
   else if (avg>=50)
   printf("\n GRADE:C");
   else
   printf("\n GRADE:D");
   }
   else
   {
   printf("\n FAIL");
   }
   getch();
   }
```

Scanned by CamScanner

Program 52:

→ Accept old balance, transaction amount, transaction type
Calculate current balance.



Program 53:

```
main()
{
clrscr();
if(1)
{
printf("\n A");
}
printf("\n B");
printf("\n C");
getch();
}
```

output: A
B
C

if (0) then
output B
C

```
if (1)
{
printf("\n A");
printf("\n B");
printf("\n C");
}
```

output A
B
C

if (0) then
not output

```
if(1)
printf("\n A");
printf("\n B");
printf("\n C");
```

output:
same as
first one

0 → false
remaining-true

Scanned by CamScanner

Program 54:

```
main()
{
if (2.2)
{
printf("\n GOOD");
}
printf("\n NICE");
getch();
}
```

output GOOD
NICE

Program 55:

```
main()
{
if(a)
printf("\n good");
printf("\n nice");
printf("\n cool");
getch();
}
```

error
undefined symbol 'a'
in function main

Scanned by CamScanner

Program 56:

```
main()
{
if (!'A')
{
printf("\n GOOD");
}
printf ("\n NICE");
printf ("\n COOL");
getch();
}
```

'A' → 65

!65 → false

output nice
cool

Program 57:

```
main()
{
int A=0;
if (A++)
printf("\n GOOD");
printf ("\n NICE");
getch();
}
```

output : NICE

```
main()
{
int A = 0;
if (++A)
printf ("\n GOOD");
printf ("\n NICE");
getch();
}
```

output : GOOD
NICE

Program 58:

```
main()
{
int A=0;
clrscr();
if (A=123,122,989)
{
printf ("\n GOOD");
}
printf ("\n NICE %d ",A);
getch(); }
```

output: GOOD
NICE 123

Scanned by CamScanner

Program 59:

```
main()
{
int A=0;
clrscr();
if (A=123,122,0)
{
printf("\n GOOD");
}
printf("\n NICE %d", A);
getch();
}
```

output: NICE 123.

Program 60:

```
main()
{
int A=0;
clrscr();
if(1);
{
printf("\n GOOD");
printf("\n NICE %d", A);
}
getch();
}
```

if(1); → null statement → which has no meaning

it will execute.

Scanned by CamScanner

→ switch-case :

- It is an alternative for if-elseif
- It is also known as bidimensional conditional statement. or Multibranching decision making statements.
- The switch-case is meant for developing Menu Driven programs.

SYNTAX:

```
switch (<var|exp|val>)
{
 case value:
    statement(s);
 case value:
    statement(s);
    ⋮
 case n:
    statement(s);
 [default:] → optional
    statement(s);
}
```

- In switch case the compiler will ensure is any case value will is match with the given number or choice
- If any case value match with the given number and the control jumps to the particular case and execute the statements associated with it and comes out
- If any case value is not match with the given number the control jumps to the default and executes the statements associated with the default and comes out.
- As per the programmer convinency we can change the order of cases. If required we can also change the order of default.

Scanned by CamScanner

- If we change order of default then break is suggested to write.
- If necessary we can also write the switch case inside the switch case.

→ **Break:**

- It is a statement used to terminate the flow of a program statements and pass out to the following statements
- It is used to comes out from the inner most block.
- It is recommended to use while, do while, for and switch case.

Program 61:

```
main()
{
int n;
printf("enter a number");
scanf("%d", &n);
switch(n)
{
case 1: printf("one"); break;
case 2: printf("two"); break;
case 3: printf("three"); break;
default: printf("Invalid");
}
}
```

Program 62:

```
main()
{
float a = 2.2;
switch(a)
{
case 1.1: printf("\nA"); break;
case 2.2: printf("\nB"); break;
}
printf("\n GOOD");
```

compilation error:
constant expression required in function main
↳ case value should not be a float.

Scanned by CamScanner

Program 63:

```
main( )
{
float a=2,b=3,c=5,
 switch(c)
 {
 case a+b: printf("\n A"); break;
 case a*b: printf("\n B"); break;
 }
  printf("\n GOOD");
 }
```

compilation error:
constant expression required in the function main.
case value should not be an expression with variables

Program 64:

```
main( )
{
 float a=2,b=3,c=5;
  switch(c)
  {
  case 2+2+1: printf("\n A"); break;
  case 3*2 : printf("\n B"); break;
  case (2+2+3)-2: printf("\n C"); break;
  }
  printf ("\n GOOD");
 }
```

compilation error:
case value should not be duplicate

NOTE:
case value must be either integer constant or character constant

Scanned by CamScanner

Program 65:

Accept a character from the user, print male if the given character is 'M' or 'm'. Print female if the given character is 'F' or 'f'. else invalid.

Menu driven programs:

Program66:

```
#include <stdlib.h>
#include <stdio.h>
# include <conio.h>
main()
{
 int ch;
 clrscr();
    printf ("\n Main Menu");
    printf ("\n ========");
    printf ("\n 1. plus \n 2. minus");
    printf ("\n 3. stars \n 4. exit");
    printf ("\n= = = = = = = =");
    printf ("\n enter your choice :");
    scanf ("%d", &ch);
    switch(ch)
     {
      case 1 : printf("\n + + + + + +");
               break;
      case 2: printf ("\n ---- --");
               break;
      case 3: printf ("\n * * * * * *");
               break;
      default :
        printf ("\n Invalid");
     }
    getch();
```

output

```
Main Menu
= = = = = = =
1. plus
2. minus
3. stars
4. exit
= = = = = = = =
Enter your choice:
1
```

Scanned by CamScanner

→ exit is a function used to terminate execute function
It belongs to stdlib.h and process.h.

Program 67:

Accept two numbers from the user. and perform the following operation

```
#include<stdlib.h>
main()
{
int a,b,ch;
printf("\n enter 2 numbers");
scanf("%d %d", &a,&b);
printf("\n Main Menu");
printf("\n----------");
printf("\n 1.ADD(+)\n 2.SUB(-)");
printf("\n 3- MUL(*) \n 4. EXIT(!)");
printf("\n --------");
printf("\n Enter your choice :");
scanf(" %d", &ch);
    switch(ch)
    {
    case1:
      print("\nADD is:%d", a+b);
      break;
    case2:
    if(a>b)
    printf("\n SUB is : %d", a-b);
    else
    printf("\n SUB is : %d", b-a);
    break;
```

```
Enter 2 numbers
Main Menu
----------
 1. ADD(+)
 2. SUB(-)
 3. MUL(*)
 4. EXIT(!)
----------
Enter your choice:
```

Scanned by CamScanner

```
case 3:
    printf ("\n MUL is: %d ", a*b);
    break;
    case 4:  exit(1);
    default:
        printf ("\n Invalid");
  }
}
```

Program 68:

Accept a number or year from the user and perform the following operator.

```
Enter no/year

Main Menu
========
1. odd/even
2. Leap or not
3. +ve/-ve
4. Exit
========

Enter your choice:
```

Scanned by CamScanner

Control Statements

When ever you want to execute a statement or group of statements repeated number of times, we need to make use of control statements or looping statements or Iterations.

control statements - 2 types

(i) unconditional control statements

eg: goto

(ii) Conditional control statements

eg. while, do-while, for

goto:

It is a statement used to transfer the flow of the program from one location to another location.

SYNTAX:

```
<Identifer>:
 statement(s);
 - - - -
 - - - - -
 goto <Label>;
```

any identifier end with colon is known as a label

In a program we can declare 'n' number of labels.

A label without goto is meaningless.

A label can be either forward or backward direction.

Program 69:

```
main()
{
MYNAME :
  printf("\n Hello");
  printf("\n chinni");
 goto MYNAME;
}
```

Scanned by CamScanner

Program 70:

```
#include <stdlib.h>
main()
{
goto NAME;
ADDRESS :
  printf("\n MIG-224");
  printf("\n KPHB");
  printf("\n HYD");
  exit(1);
 NAME :
    printf("\n SSSIT");
    printf("\n Computer education");
  goto ADDRESS;
 }
```

output:

```
SSSIT
computer education
MIG-224
 KPHB
HYD
```

Program 71:

```
main()
int count;
  count=1;
 My Name :
  if (count ==4)
   exit(1);
  else
   printf("\n Chinni");
   count++;
 } goto MyName;
```

Scanned by CamScanner

Program 72:

```
# include <stdlib.h>
main( )
{
int a=2;
clrscr( );
switch(a)
{
case 1 : printf ("\n A"); break;
case2 : printf ("\n B"); exit(1);
}
Printf("\n GOOD");
goto case2;
getch();
}
```

output:

GOOD

B

case1 - label

case 1 → switch case.

→ **While loop:**

SYNTAX:

```
<exp1>; initialisation
while <epx2> condition
 {
 statement(s);
 <Exp 3>; increment/decrement
 }
```

In the while loop it will check the condition initially, if the condition evaluates to true it will enter into the loop and executing the looping statements till the condition gets false.

While loop is called an "entry level looping statement".

Scanned by CamScanner

hint:

	Exp 1	Exp 2	Exp 3
1 to 10	1	< & <=10	++
10 to 1	10	> & >=1	--

→ Program 73:

```
main( )
{
 int i;
 i=1;
 while (i<=3)
 {
 printf("\n Hello");
  i++;
  }
}
```

output : Hello
Hello
Hello

→ Program 74:

To print five stars in vertical

```
main ()
{
int i;
 i=1;
while (i<=5)
 {
 printf ("\n * ");
   i++;
  }
}
```

output: *
*
*
*
*.

Scanned by CamScanner

→ **Program 75:**

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

output: 1
2
3
4
5

→ **Program 76:**

```
main()
{
int i;
 i=10;
while (i>=5)
 {
 printf("\n%d",i);
  i--;
 }
}
```

output: 10
9
8
7
6
5

→ **program 77:**

Program to generate even numbers from 2 to 10

```
main()
{
int i;
 i=2;
while(i<=10)
{
printf("\n %d", i);
 i=i+2;
}
```

output: 2
4
6
8
10

Scanned by CamScanner

→ **Program 78:**

Accept a number from the user, generate the number upto given number

```
main()
{
 int i,n;
 printf(" Enter a number");
 scanf(" %d", &n);

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

   i++;
   }
 }
```

sd

1 0 0 0

1 0 0

→ **Program 79:**

Accept a number from the user, generate squares series from 1 to given number

```
main()
{
int i,n;
printf(" enter a number");
scanf(" %d", &n);
 i=1;
 while (i<=n)
 {
 printf(" \n %d its square is %d", i, i*i);

   i++;
   }
 }
```

out put:

enter the number = 3

1 its square is 1

2 its square is 4

3 its square is 9

Scanned by CamScanner

→ Program 80:

Accept a number from the user and generate the table for the given number.

```
main()
{
 int i,n;
 printf (" enter the number");
 scanf ("%d", &n);
 i=1;
while(i<=10)
{
printf ("\n %d x %2d = %-3d", n,i, n*i);
 i++;
 }
}
```

→ Program 81:

Program to find out sum of natural numbers

```
main()
{
int i,n,sum=0;
printf (" enter the number");
scanf (" %d", &n);
 i=1;
 while (i<=n)
 {
   sum=sum+i;
   i++;
 }
 printf ("\n sum=%d", sum);
 }
```

Scanned by CamScanner

→ **Program 82:**

Accept a number from the user. Find out the factorial of a given number

```
main()
{
long int i,n,fact=1;
Printf("enter the number");
scanf("\n %ld ", &n);
 i=1
while(i<=n)
{ Fact=Fact*i;
  i++;
}
Printf("\n Fact : %ld", &Fact);
}.
```

→ **Program 83:**

Accept a number from the user and find out the number of digits in the number

```
main()
{
int n;
int r, nd=0;
printf ("enter the number");
scanf ("\n%d", &n);
while (n>0)
{
 r=n%10;
  nd++;
 n=n/10;
}
printf("\n no. of digits= %d", nd);
}
```

Scanned by CamScanner

→ Program 84:

Accept a number from the user and find out sum of digits of the given number

```
main()
{
int n;
int r, sum=0;
printf("\nenter the number");
scanf("%d", &n);
while(n>0)
{r= n%10;
sum=sum+r;
n=n/10;
}
Printf(" sum of the given number= %d", sum);
}
```

→ Program 85:

Accept a number from the user and print the given number in reverse

```
main()
{
int n;
int r, rev=0;
printf("\n enter the number"),
scanf(" %d", &n);
while(n!=0)
{
r= n%10;
rev=(rev*10)+r;
n=n/10;
}
printf(" reverse of the number = %d", rev);
}
```

Scanned by CamScanner

→ **Program 86:**

Accept a number from the user and find out whether the pro number is a palindrome or not.

```
main()
{
int n;
int r, rev=0, temp;
printf("\nenter the number");
scanf("%d", &n);
temp=n;
while(n!=0)
{ r=r%10;
rev= rev*10+r;
n=n/10;
}
if (temp==rev)
printf("given number is a palindrome");
else
printf("given number is not a palindrome");
}
```

→ **Program 87:**

Accept a number from the user and first out first and last digit of in the given number

→ **Program 88:**

Accept a number from the user and find out the given number is Amstrong or not

$n=153$ $1^3+5^3+3^3 \rightarrow 153 \rightarrow$ amstrong number.

```
int sum = sum + (r*r*r);
```

Scanned by CamScanner

→ do---- while:

SYNTAX:

```
<exp1>;
do
{
statement(s);
<exp3>;
}
while(<exp2>;
```

In the do-- while it will execute the statement(s) for atleas one time before checking the condition and execute the looping statements, till the condition is false.

Hence do-while is an exit level looping statement.

→ Program 89:

```
main()
{
int i;
i=16;
do
{
printf ("\n %d",i);
i++;
}
while (i<=3);
}
```

output:

16

Scanned by CamScanner

→ **Program 90:**

Program to generate even numbers from 1 to 10

```
main( )
{
int i;
clrscr( );
 i=2;
do
{
printf("\n%d", i);
i=i+2;
}
while(i<=10);
getch( );
}
```

```
main( )
{
int i;
clrscr( );
i=2;
do
{
if (i%2==0)
{ printf ("\n%d", i);}
i++;
}
while (i<= 10);
getch( );
}
```

→ **Program 91:**

Accept a number from the user and find the ~~p~~factorial of the given number.

```
main( )
{
int i, n, fact=1;
printf (" enter a number");
scanf ("%d", &n);
if (n==0)
{ printf ("\n Fact is %d", fact); }
else
 {
   i=n;
   do
   {
   fact=fact*i;
   }  i--;
```

Scanned by CamScanner

```
while (i>=1);
printf ("\n Factorial is %d", fact);
}
}
```

→ Program 92:

Accept a number from the user and print factors of the given number.

```
main()
{
int i,n;
printf("\n enter the number");
scanf("%d", &n);
i=1;
do
{
if (n%i==0)
{printf("\n %d", i);}
i++;
while (i<=n);
}
```

→ Program 93:

Accept a number from the user and find out sum of factors of the given number.

main()

→ Program 94:

Accept a number from the user and find out the given number is prime or not

Scanned by CamScanner

```
main()
{
int i, n, count=0;
  f=1;
printf("enter the number");
scanf("%d", &n);
do
{
  if (n%i==0)
   {count++;}
 } i++;
while(i<=n);
if (count==2)
printf("\n Prime");
else
printf("\n not a prime");
```

→ Program 95:

Accept a number from the user find out the given number is perfect or not

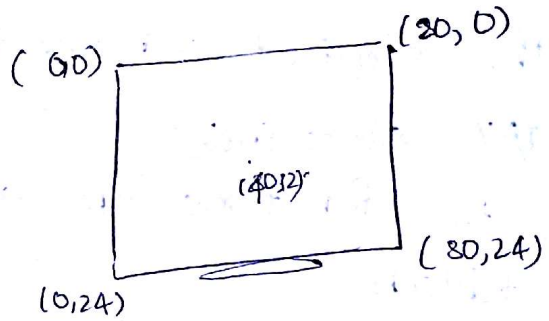
```
main()
{
int i, n, sum=0;
 i=1;
Printf("enter the number");
scanf("%d", &n);
do
{ if (n%i==0)
  sum= sum+i;
 }
 i++;
while (i<n);
```

Scanned by CamScanner

```
printf("sum of factors = %d", sum);
}
if (sum==n)
printf(" The given number is perfect");
else
printf(" The given number is not perfect");
}
```

→ **gotoxy():**

It is used to locate the cursor at the specified location on the screen. Usually every monitor has 24 rows and 80 colums.



SYNTAX:

```
gotoxy(int,int);
```

eg: goto xy(40,12);
printf("Hello");

→ **Textcolour:**

It will produce the specified text with the specified colour.

SYNTAX:

textcolor(COLORNAME / code);

eg, text color(14); (or) textcolor(YELLOW);

→ **text background:**

It will produce a background colour for the specified text

SYNTAX: textbackground (COLORNAME/code),

e.g., text background(14);

Scanned by CamScanner

→ while working with this two for we should use 'cprintf' for rather printf

cprintf – colour print format

→ This functions are supported by stdio.h header file

→ delay():

It is used for delaying the execution of the programming statements till the specific statemen time in now seconds.

SYNTAX: delay(long);

→ sleep():

It will delay the execution of programing statements till the specified is seconds

SYNTAX: sleep(int);

→ These two functions are seperated by dos.h header file (or) stdlib.h

→ Progemas:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
goto(xy)(4,8)
text to color 8(w)
text blacking 4%
cprintf("Hellow");
getch()
```

Scanned by CamScanner

→ Program 97:

```
#include <stdio.h>
#include<< dos.h>
{
int l;
clrscr();
l=1;
do sleep(dil>
gotoxy gotoxy (23,0);

textowlour T
```

→ Program 98:

```
#include< stdio.h>
#include <dos.h>
main( )
{
int x;
clrscr();
x=1;
do
{
delay( 100000000);
clrscr();
gotoxy (x,12);
textcolor (1);
cprintf ("chrnn1");
x++;
} while (x<=50);
getch();
```

Scanned by CamScanner

Program 99:

```
#include <stdio.h>
#include <dos.h>
main()
{
int x1, x2;
clrscr();

x1=1 ; x2=50;
do
{
delay (100000000);
clrscr();
goto xy(x1, 8); Printf ("sai");
gotoxy (x2,10); printf ("Baba");
x1++ ; x2--;
} while (x1<=26);

getch();
}
```

→ for loop :

```
for ([exp1]; [exp2]; [exp3])
{
statement(s);
}
```

→ for loop will work exactly with while loop. First for loop is also an entry level looping statement.

→ one and only the looping statement which is executed without having any condition.

Scanned by CamScanner

In the while and do-while, the condition is manditary

Program 100:

```
main()
{
int i;
for (i=1; i<=3; )
{ Printf ("\n %d", i);
  i++; }
}
```

```
main ( )
{
int i=1;
for (; i<=3; i++)
{ Printf ("\n %d", i);}
}
```

output: 1
2
3

Program 101:

```
main()
{
int i;
for (i=1; i<=3; i++)
{printf ("\n %d", i); }
}
```

output: 1
2
3

program 102:

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

output: 1
2
3
1
3276 7

program 103:

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

output: 1
1
1
1

Program 104:

```
main()
{
int i=20;
for (i=20; i<=25; i++) (i) -> continue
{ printf("\n %d", i);}
}
```

output: 24

Scanned by CamScanner

Program 105:

```
main( )
{
for( ; ; )
  ;
}
```

output:

No output

compilation success

program 106:

```
main( )
{
int i=10;
for( ; i<=20)
  {;}
pf ("\n%d",i);
}
```

Compilation error

expression missing in syntax

(exp-3)

program 107:

```
main( )
{
int i,j;
clrscr();
for (i=1,j=5; i<=5; i++,j--)
  printf("\n %d \t %d", i, j);
getch();
}
```

output.

1	5
2	4
3	3
4	2
5	1

Program 108:

output:

1	5
SSSIT	KPHB
3	3
SSSIT	KPHB
5	1

Scanned by CamScanner

Nested loops:

Based on the programming requirements we can use a looping statement inside another looping statement

```
for (exp1; exp2; exp3)
{
  for (exp1; exp2; exp3)
   {
    statement(s);
   }
}
```

Program 109:

```
main()
{
int o,i;
for (o=1; o<=3; o++)
  {
    for(i=1; i<=2; i++)
    { printf("\n sai"); }
     printf("\n");
  }
}
```

output:

```
sai
sai

sai
sai

sai
sai
```

Program 110:

```
main()
{
int o,i;
for (o=1; o<=3; o++)
  {
   for (i=1; i<=3; i++)
    { printf(" * "); }
     printf("\n");
  }
```

output:

```
* * *
* * *
* * *
```

Scanned by CamScanner

Program 111:

```
main()
{
 int o,i;
 for(o=1; o<=3; o++)
  {
   for (i=1; i<=3; i++)
    { printf(" %3d",o); }
    printf("\n");
   }
 }
```

output:

```
1 1 1
2 2 2
3 3 3
```

Program 112:

output

```
1 2 3
1 2 3
1 2 3
```

Program 113:

```
main()
{
int o,i;
 for (o=3; o>=1; o--)
  {
   for (i=1; i<=o; i++)
    { printf(" * "); }
    printf("\n");
  }
}
```

output:

```
* * *
* *
*
```

Scanned by CamScanner

Program 114:

1)
```
*
* *
* * *
```

2)
```
1
2 2
3 3 3
```

3)
```
1
1 2
1 2 3
```

4)
```
3 2 1
2 1
1
```

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

6)
```
*
* *
* * *
* * *
* *
*
```

7)
```
1
* *
3 3 3
* * * *
5 5 5 5 5
```

8)
```
1
1 *
1 * 3
1 * 3 *
1 * 3 * 5
```

9)
```
*
    *
        *
```

10)
```
00 01 02
10 11 12
20 21 22
```

Program 115:

Accept a number from the user, generate tables series for the gi up to the given number

```
main()
{clrscr();
int n,i,j;
printf(" enter the value of n");
scanf (" %d ", &n);
for (j=1; j<=n; j++)
{
for (i=1; i<=10; i++)
{ printf(" %d * %3d = %3d", j, i, j*i); }.
printf("\n");
printf (" press any enkey to continue--");
getch();
} clrscr();
getch();
}
```

Scanned by CamScanner

→ Program 116:

```
main()
{
clrscr();
int h,m,s;
while(1)
{
  for (h=0; h<=23; h++)
   {
   for (m=0; m<=59; m++)
    {
     for(s=0; s<=59; s++)
      sleep(1);
      clrscr();
      gotoxy (22,8);
      printf("%d : %d : %d", h,m,s);
     }
   }
 }
}
```

Scanned by CamScanner

Arrays

→ An ordinary variable can store a value.

→ For the ordinary variables, memory allocations are random m positions

→ An array is a memory allocation which enables you to store 'n' no of homogenous mixture of elements and share the variable name

→ For the array variables, memory allocations are consequent locations, hence reading the data from the array variable is faster than an ordinary variable

→ Arrays are classified into 3 types

1. single-dimensional array (SDA or 1-Darray)
2. double-dimensional array (DDA or 2-Darray)
3. multi-dimensional array (MDA or 3-Darray)

→ <u>Single-dimensional array</u>:

It is a memory allocation which enables you to store 'n' no. of homogenous mixture of elements in a linear fashion (sequentially)

→ While working with the arrays, we must specify the size of an array before compilation either in direct subscription or indirect subscription

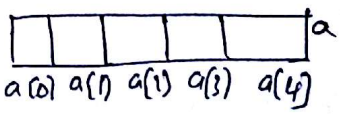
→ If we enter the data less than the specified capacity, there is a possibility of wastage of memory in direct subscription

→ Once, we define the size of an array, it doesnot allow you to increase or decrease the size of an array

→ Array index value always starts with '0' and ends with (size - 1)

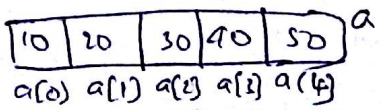
syn: <datatype> <vname> [size];

e.g: int a[5];



<u>Assign values</u>:

int a[5] = {10, 20, 30, 40, 50};



a(0) = 10 a(1) = 20
a(2) = 30 a(3) = 40
a(4) = 50

Scanned by CamScanner

int a[5] = {10,20,30}

10	20	30			a
a(0)	a(1)	a(2)	a(3)	a(4)	

→ Printing values :

Pf ("\n %d", a[0]); 10

Pf ("\n %d", a[1]); 20

Pf ("\n %d", a[4]); 50

Pf ("\n %d", a[2]); 30

Pf ("\n %d", a[3]); 40

i.e., for (i=0; i<5; i++)

{ Pf ("\n %d", a[i]); }

→ int a[5]; // valid

int a[0]; → invalid

int a[-125]; → invalid

→ int n=5;

int a[n]; // invalid

int a[A]; // invalid

int A['a']; a → 97

i.e., int A [97] // valid

→ int a =10;

int a[a]; // invalid

→ int a[3] = { , , 30};

invalid

→ int a[3] = {10}; valid

→ int a[3] = 10; // invalid

→ int a[3] = {0,0,30}; // valid

Scanned by CamScanner

→ **Program 117:**

Program for printing array elements

```
main()
{
 int a[5] = {10,20,30,40,50},i;
  clrscr();
 for (i=0; i<5;i++)
  { printf("\n a[%d] = %d", i,a[i]);
     sleep(1)}
  Printf("= = = = = = ");
 for (i=4; i>=0;i--)
 {printf ("\n a[%d]= %d", i,a[i]);
    sleep(1);}
  getch();
}
```

output

```
a(0) = 10
a(1) = 20
a(2) = 30
a(3) = 40
a(4) = 50
= = = = = = =
a[4] = 50
a[3] = 40
a[2] = 30
a[1] = 20
a[0] = 10
```

→ **Program 118:**

accept 5 elements from the user and print it on the screen

```
main()
{
int a[5],i, sum=0;
 clrscr();
for (i=0;i<5;i++)
printf (" enter the element @ a[%d]=", i);
scanf (" %d", &a[i]);
for (i=0; i<5;i++)
{ sleep(1);
printf ("\n %d", a[i]);
 sum=sum+a[i]; }
printf(" sum of array : %d", sum);
getch();
}
```

output

```
enter the element @ a[0]=10
 "   "   "  @ a(1) =20
 "   "   "  @ a(1) =30
 "   "   "  @ a(3) =40
 "   "   "  @ a(4) =50
10
20
30
40
50
sum of array :. 150
```

Scanned by CamScanner

→ Program 119:

Accept s elements from the user and print even elements from it

```
main( )
{
int a[s], i; sum
printf("\nenter s elements");
for ( i=0; i<s; i++)
  { scanf ("%d", &a[i]);}
printf ("\n even numbe elements in the list");
for (i=0; i<s; i++)
  {
   if (a[i]%2==0)
   { printf ("\n %d", a[i]); }
   }
}
```

→ Program 120:-

Accept s elements from the user, find out avg of odd and even elements

```
main( )
{
int a[s], i, no, ne, so, se, a;
float ae, ao;
      no=ne=so=se=0
for (i=0; i<s; i++)
{ printf ("\n enter an element");
   scanf ("%d", &a[i]); }
for (i=0; i<s; i++)
 { if (a[i]%2==0)
     { se = se+a[i];
       ne++;
     }
```

Scanned by CamScanner

```
else
 {
    so= so+a[i];
      no++;
    }
 }
 ae= se/ne;
 ao= so/no;
 printf(" so=%d n no=%d ";so,no);
 printf(" se=%d n ne=%d";se,ne);
 printf(" ae= %0.2f n ao= %0.2f"; ae,ao);
 }
```

→ <u>Program 121:</u>

Accept s elements from the user and ask an element for search and find out the existency of the search element

```
main( )
{
int a[s], sn, i, found=0;

for (i=0; i<s; i++)
 { printf("\n enter the element");
   scanf("%d ", &sn);
 }
for (i=0; i<s; i++)
  {
   if (a[i]==sn)
    { found =1;
          break; }
  }
(found==1)? printf("\n found"): printf("\n not");
}
```

Scanned by CamScanner

→ **Program 122:**

Accept 5 elements from the user, find out biggest, smallest element in the array.

```
main()
{
int a[5], big, small, i;

clrscr();

for(i=0; i<5; i++)
  { Printf("\n enter an element : %d");
    scanf(" %d", &a[i]); }

  big= a[0];
  for (i=1; i<5; i++)
   { if (a[i]>big)
       big=a[i];
   }
   Printf ("\n biggest is : %d", big);

  small= a[0];
  for (i=1; i<5; i++)
   { if (a[i]<small)
       small= a[i];
   }
   Printf ("\n smallest is: %d", small);

  big= small=a[0];
  for (i=1; i<5; i++)
   { if (a[i]>big)
       big=a[i];
     else if(a[i]<small)
       small= a[i];
   }

   Printf (" biggest : %d", big);
   printf (" smallest : %d", small);
   getch();
}
```

biggest

smallest

biggest and smallest

Scanned by CamScanner

→ **Program 123:**

Accept 10 elements from the user, ask an element for search. Find out occurency of search number

→ **Program 124:**

Accept 10 elements from the user, ask an element for delete and delete the element from the list if found.

→ **Double dimensional array:**

→ It is an array allocation, which enable you to maintain and manipulate the data element in the combination of rows and columns. Simply, in the form of the matrix

→ syn: <datatype> <vname>[size][size];

eg: int a[2][2];

Assign values:

```
int a[2][2] = { {10,20}, {30,40} };
int a[2][2] = { 10,20,30,40};
int a[2][2];
    a[0][0] = 10; a[0][1] = 20;
    a[1][0] = 30; a[1][1] = 40;
```

	0	1
0	a[0][0] 10	a[0][1] 20
1	a[1][0] 30	a[1][1] 40

a

Printing values:

```
printf("%d", a[0][1]); 10
printf("%d", a[0][1]); 20
printf("%d", a[1][0]); 30
printf("%d", a[1][1]); 40
```

```
for (int r=0; r<2; r++)
{
  for (int c=0; c<2; c++)
  {
    printf("%d", a[r][c]);
  }
  printf("\n");
}
```

Scanned by CamScanner

Program 125.

Program to print array elements along with its index

```
main()
{
int r, c;
int a[2][2] = {{10,20}, {30, 40}};

for (r = 0; r<2; r++)
  {
   for (c=0; c<2; c++)
    {
     sleep(1);
     printf ("a[%d][%d] = %d", r, c, a[r][c]);
    }
   printf("\n");
  }
}
```

program 126:

Accept a 2x2 matrix and print transpose matrix

```
main()
{
int r,c,a[2][2];

for (r=0; r<2; r++)
 {
  for (c=0; c<2; c++)
   {
    printf("\n enter an element  a[%d][%d] : ", r,c);
    scanf (" %d",&a[r][c]);
   }
 }
 printf("\n \n");
```

Scanned by CamScanner

```
for (r=0; r<2; r++)
 {
  for (c=0; c<2; c++)
   {
    sleep(1);
    printf (" %5d", a[c][r]);
   }printf("\n");
 }
```

Program 127:

Accept a 3x3 matrix and find out row wise calculations

```
main()
 {
  int a[3][3], r, c, sum;

  for (r=0; r<3; r++)
   {
    for (c=0; c<3; c++)
     {
      printf("enter an element  a[%d][%d] : ", r, c);
      scanf (" %d", a[r][c]);
     }
   }
   for (r=0; r<3; r++)
    { sum=0;
     for (c=0) c<3; c++)
     {
      printf (" %6d ", a[r][c]);
      sum= sum+ a[r][c];
     }
     printf (" =%5d ", sum);
     printf(" \n");
 }
}
```

Scanned by CamScanner

→ **Program 128:**

Accept a 3x3 matrix, find out column wise calculations

→ **Program 129:**

Accept a 3x3 matrix, find out biggest element of daigonal elements

```
main()
{
int a[3][3],r,c,big;
  for (r=0; r<3;r++)
   {
   for (c=0; c<3;c++)
    {
    printf("\nenter an element a[%d][%d] : ", r,c);
    scanf (" %d", &a[r][c]);
    }
   printf("\n");
  }
  printf("\n");
  big= a[0][0];
  for (r=0;r<3;r++)
  {
   for(c=0;c<3;c++)
    {
     if (r==c)
     {
     /* printf ("%5d", a[r][c]); */
     if (a[r][c] > big)
     { big= a[r][c]; }
     }
    }}
  printf ("\nbiggest is %d ", big);
```

Scanned by CamScanner

→ Program 130:

Accept 2 2x2 matrix, find out addition of two matrices

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int a[2][2], b[2][2], s[2][2], r,c;
clrscr( );
/* values of a[][] */
for (r=0; r<2; r++)
  {
   for (c=0; c<2; c++)
    {
    Printf ("\nenter the elements @ a[%d][%d] ", r,c);
    scanf ("%d", & a[r][r]);
    }
  }
/* values of b[][]* /
for (r=0, r<2; r++)
  {
   for (c=0; c<2; c++)
    {
    printf ("\n enter the elements @ b[%d][%d]", r, c);
    scanf ("%d ", &a[r][c]);
    }
  }
/* Add of a[][] + b[][]* /
for (r=0; r<2; r++)
{
  for (r=0 ; c<2; c++)
   {
   s[r][c] = a[r][c]+b[r][c];
  }
}
```

Scanned by CamScanner

```
/* printing values of s Array */
for (r=0; r<2; r++)
{
 for (c=0; c<2; c++)
  {
   printf ("%5d", s[r][c]);
  }
 printf ("\n");
}
getch();
}
```

→ **Program 13p:**

Accept two 2x2 matrices. Find the product of two matrices.

→ **Multi Dimensional Array:**

It is a memory allocation which enable you to maintain (store) and manipulate the the data elements in the combination of tabbles, rows and columns

Syn: <datatype> <varname> [size][size][size];

→ **Program 132:**

```
#include <stdio.h>
#include <conio.h>
main( )
{
 int a[3][2][2] = {
                  { {10,20}, {30,40} },
                  { {50,60}, {70,80} },
                  { {10,40}, {60,40} }
                  };
 int t, r, c;
```

Scanned by CamScanner

```
clrscr();
for (t=0; t<3; t++)
{
 printf ("\n elements in a[%d] table \n ", t);
  for (r=0; r<2; r++)
   {
    for (c=0; c<2; c++)
     {
      printf ("%5d", a[t][r][c]);
      }
     printf ("\n");
    }
  }
 getch();
}
```

Scanned by CamScanner

Strings

→ string is a character array which enable you to store a single character or group of characters. The compiler implicitly which place null character ('\0') at the end of each string

→ Null is not a printable character i.e, it wont be appear on the screen and won't print on the paper.

→ '\0' is not a space, not a digit, not a sysmbol but it will take 1 byte space in the memory

→ syn:

<data type> <name> [size];

Eg; char s[5];

Assignatives:

Eg, ~~char s[] = "sai\0";~~

~~char s[5] = {"sina"};~~

A3 chylo Rs

s	a	i	\0	
s(0)	s(1)	s(2)	s(3)	s(4)

Assign values:

Eg: char s[5] = "sai", or

char s[5] = {"sai"}; or

char s[5] = {'s' 'a' 'i'};

printing values:

printf("%c", s[0]); s

printf("%c", s[1]); a

printf("%c", s[2]); i

printf("%c", s[3]);

printf("%c", s[4]); sai

s

printing:

1) for (i=0; i<100; i++)
{ pf("%c", s[i]); }

2) for (i=0; i<100; i++)
{ if (s[i] != '\0')
printf("%c", s[i]);
else
break;

3) for (i=0; s[i] != '\0'; i++)
{ printf("%c", s[i]); }

Scanned by CamScanner

→ **NOTE:**

When a string is not properly terminated at the time of printing the string you will get some garbage value along with the string

→ **Program 133:**

```
main()
{
char s[4]= "baba";
clrscr();
printf ("%s", s);
getch();
}
```

output: baba garbage

→ **Program 134:**

```
main()
{
char f[3]= "sai", s[5]= "baba";
clrscr();
printf ("\n %s \n %s", f, s);
getch();
}
```

sai baba
baba

→ **Program 135:**

```
main()
{
char s[]= {97,122, 90,66};
clrscr();
printf ("%s", s);
getch();
}
```

a z ZB garbage value

Scanned by CamScanner

Program 136:

```
main()
{
char s[5]= {A,B,a,b,c};
clrscr();
printf("%c", s[2]);
printf("%c", s[4]);
getch();
}
```

error.

→ Regular expressions:

reading the data from the user based on a specific pattern called regular expressions

Program 137:

```
main()
{
char s[200];
clrscr();
printf("\n enter your data");
scanf("%[A-Z]", &s);   (it will read only capital alphabet)
scanf("%[A-Z][a-z]", &s);  (it will read only alphabet)
scanf("%[A-Z, a-z, 0-9]", &s);  (it will read a numerical values)
scanf("%^[A-Z, a-z, 0-9]", &s);  (it will read only symbols)
scanf("%[M,m,f,F]", &s);  (it will read only either M,m,f,F symbols)
scanf("%[^\n]", &s);  (it will read all character till user press enter)
scanf("%[^\0]", &s);  (it will read all char till user press F6)
printf("Given data is: %s", s);
getch();
}
```

Scanned by CamScanner

→ getch();

It is an input function waiting for getting the character from the console without echoing and processed it for immediately

→ getche():

It is also an input function waiting for getting the character from the console with echoing and processed it for immediately

→ getchar():

It is also an input function waiting for getting a char from the console with displaying the processed after pressing enter.

→ Program 138:

```
#include<stdio.h>
#include<conio.h>
main().
{
 char x;
 clrscr();
 printf("enter anly char : ");
   scanf("%c", &x);
   x = getch();
   x = getche();
   x = getchar();
  printf("\n given char is : %c", x);
  sleep(4);
}
```

any of these can be used.

Scanned by CamScanner

→ NOTE:

While extracting the string from the user while using scanf it will take only first word of the given statement

→ fflush():

It is used to erase or flush the screen (It will eraser all non-printable characters which are occuring through standard input)

→ Program 139:

```
#include<stdio.h>
main()
{
char name[20], g;
clrscr();
printf ("\n enter name");
scanf ("%s", &name);
fflush (stdin);
printf ("\n enter gender");
scanf ("%c", &g);
printf ("\n Name is : %s", name);
printf ("\n Gender is : %c", g);

getch();
}
```

→ Input and output functions:

- input and output functions are used to read the values from the user and assign to the variable.
- output functions are used to display the msg on the console. These are also used to read the values from the variables and display it on the screen

Scanned by CamScanner

- Input and output are classified into two categeories

(i) formatted I/O functions:

- These are suitable for any type of data
- It will support escape sequences
- It will support format specifiers (e.g scanf() fscanf(), printf(), cprintf())

(ii) unformatted I/O functions:

- It is suitable for only character types
- It will support escape sequences
- It doesnt support format specifiers
- e.g, getc(), putc(); puts(), gets()

→ Program 140:

Accept a strong from the user and print the given string in vertical

```
main( )
{
 char s[5],

 int i;
 clrscr();
 puts("enter any string");
 gets(s);
  for (i=0; s[i]!='\0'; i++)
  {printf ("\n %c", s[i]);}

 getch();
 }
```

Scanned by CamScanner

→ Program 141:

Accept a string from the user find out length of the string.

```
main( )
{
  char s[200];
  int i, len=0;
  clrscr();
  puts("enter any string"); gets(s);
/* for (i=0 ; s[i] != '\0' ; i++)
    {length++;}
  printf("\n" Length is %d", leng); */
  for (i=0; s[i]!= 0; i++)
  printf "\n Leave length is= %d", i);
  getch();
}
```

Program 141:

→ Accept the given string for the user and print it in reverse. reverse

```
main()  main( )
void ...  {
clrscr()   char s[200]; int i; clrscr();
  puts("enter the string");
  gets(s);
  for (i=0; s[i]! = '\0'; i++);
  for (i= i-1; i>=0; i--)
    { printf("%c", s[i]; }
  getch();
```

Scanned by CamScanner

→ Program 143:

Accept a string from the user. Copy the string from one to another.

```
main()
{
 char s[20], d[20];
 int i;
 clrscr();
 puts ("enter any string");
  gets(s);
  for (i=0; s[i] != '\0'; i++)
  { d[i]= s[i]; }
  d[i]= '\0';
  printf ("\n string after copy : %s", d);

 getch();
}
```

→ Program 144:

Accept two strings from the user, concatenate the two strings (combine)

```
main()
{
 char s[5], f(s], c[s];
  int i, j, k;
  clrscr();
  puts (" enter string");
  gets (f);
  puts (" enter string");
  gets (s);
  for(i=0, k=0, s[i]! = '\0'; i++; k++)
    { c[k]= s[i]; }
  for(j=0, f[i] != '\0'; j++; k++)
  { c[k] = f[i]; }
```

Scanned by CamScanner

```
c[i]='\0';
printf(" enter the string:%s", c[i]);
getch();
}
```

→ **Typecasting:**

- Type casting is a process of converting the value from one type to another type.
- Type casting is classified into two types
  (i) implicit typecasting
  (ii) explicit typecasting
- Implicit typecasting is a process of converting the value from one type to another type by the complier automatically which is possible integer type to character type and characterype to integer type.
- → Explicit typecasting is a process of converting the value from one type to anotha type by us.

→ **Program 145:**

```
main()
{
int x=65;
char y='A';
clrscr();
 printf("\n %d", x);
 printf("\n %d", x);
  printf("\n %c", y);
  printf("\n %d", y);
getch();
}
```

Scanned by CamScanner

→ **Program 146:**

```
void main()
{
clrscr();
   printf("\n %d", 'A'+'B');
   printf("\n %c", 33+32);
 getch();
}
```

→ **Program 147:**

```
mam()
{
 float me= 2.2;
 clrscr();
    if (me==(float) 2.2)
      printf ("\n Good");
    else printf ("\n nice");
    getch();
}.
```

→ **Program 148:**

```
mam()
{
int a=5, b=2;
floact c;
clrscr();
   c= (float)a/b;
 printf("\n result is: %.2f", c);
 getch();
}
```

Scanned by CamScanner

→ **Program 149:** Program to generate ASCII values

```
main()
{
 int i;
 clrscr();
 for (i=0; i<=255; i++)
  { printf("%d =%c", i, i);}
 getch();
}
```

65-90 = 'A' to 'Z'
97-122 = 'a' to 'z'
48-57 = '0' to '9'
32 = space
27 = escape
8 = Backspace
13 = enter

→ **Program 150:**

Accept a string from the user and convert the given string into uppercase letters.

```
main()
{
int i;
char s[20]
puts
printf("enter the string");
gets(s);
for (i=0; s[i]!='\0'; i++)
 {
   if(s[i]>=97 && s[i]<=122)
        {s[i]-=32}
   { Printf("\nstring in uppercase = %s", s);}
   }
getch();
}
```

Scanned by CamScanner

→ Program 151:

Accept a string from the user and convert the string into lower case letters.

→ Program 152:

Accept a string from the user and convert the string into toggele case format.

```
main()
{
 char s[20];
 int i;
 puts(" enter the string ");
 gets(s);
  for (i=0; s[i]!='\0';i++)
   {
      if (s[i]>=97 && s[i]<=122)
       { s[i]-=32;}
       elseif( s[i]>=65 && s[i]>=90)
        { s[i]+=32;}
      }
    printf("\n enter the string in Tcase = %s ", s);
  getch();
 }
```

output:

enter the string chinni KHANNA 1234

string in tcase CHINNI khanna 1234

→ Program 153:

Accept a string from the user, convert the string into title case format

→ Program 154:

Accept a string from the user, display the given string into abbrivated from

Scanned by CamScanner

→ Program 155:

Accept a string from the user find out the number of capitals, no. of smalls, no. of digits, no. of symbols, no of words, total no. of letters and no of vowels in the string.

```
main()
{
 char s[200];
 int i,nc,ns,nd, nsy, tnl, nv, nw;
    nc=ns=nd=nsy=tnl=nv=0;
          nw=1;
   clrscr();
   puts("enter string");
   gets(s);
   for(i=0; s[i]!='\0'; i++)
   {
     switch(s[i])
     {
      case'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
       nv++;
     }
     if (s[i] >=65 && s[i]<90)
       nc++;
     else if (s[i] >=97 && s[i]<=122)
       ns++;
     else if (s[i]==32)
       nw++;
     else if( s[i]>=48 && s[i]<=57)
       nd++;
```

Scanned by CamScanner

```
else
    nsy++;
}
tnl = nc+ns+nd+nsy;
printf("\n NC is %d \n NS is : %d", nc, ns);
printf("\n ND is %d \n NSY is : %d", nd, nsy);
printf("\n NW is %d \n TNL is : %d", nw, tnl);
Printf("\n NV is %d", NV);
getch();
}
```

→ Strings with double dimensional arrays

It is the process of representing or storing group of strings in a variable

syn : <datatypes> <vname> [size][size];

eg. : char s[3][10] = {"sai", "chinni", "Ambica"};

	0	1	2	3	4	5	6	7	8	9
0	s	a	i	\0						
1	c	h	i	n	n	i	\0			
2	A	m	b	i	c	a	\0			

printf("%s", s[0]); Sai

Printf("%s", s[1]); chinni

Printf("%s", s[2]); Ambica

Scanned by CamScanner

→ Program 156:

```
main()
{
char s[3][20] = { "1. Ramesh", "2. Mahesh", "3. Suresh"};
int i;
clrscr();
for (i=0;i<3;i++)
    {
      sleep(2);
      clrscr();
      gotoxy (20,6);
      printf ("%s", s[i]);
    }
getch();
}
```

→ Program 157: (QUIZ)

```
main()
{
char q[3][100] = { "1. How many keys on mouse",
                   "2. Who is CM of AP",
                   "3. Who is Father of c"};
cha m[3][100] = { "a. 5   b. 2   c. none",
                  "a. YSJ  b. NCB  c. KCR",
                  "a. Pitche  b. James  c. Nani"};
char key[3] = { 'c', 'b', 'a'};
char ans, choice;
int marks=0, i;
```

Scanned by CamScanner

```
clrscr();
gotoxy (20,6); textcolor (10);
                printf ("WELCOME");

gotoxy (23, 7); textcolour (14);
                cprintf ("TO");

gotoxy (20,8); textcolor (15);
                cprintf("QUIZ WORLD");

gotoxy(18,10); textcolor(15);
                cprintf ("R. u ready Y/N :");
                scanf ("%c", &choice);

  if (choice =='y' ||choice =='Y')
  {
    for (i=0; i<3; i++)
    {
      clrscr();
      gotoxy (20, 7); text color (14);
       cprintf ~~cprintf~~ ("%s", q[i]);

       gotoxy (20,9); text color (15);
        cprintf("%s", m[i]);

       goto (20,11); text color (10);
        fflush (stdin);
        printf ("Enter Ans: ");
        scanf ("%c", &ans);
            if (ans ==key[i])
                {marks +=10;}
      }
```

Scanned by CamScanner

```
    sleep(1);
    clrscr();
    gotoxy(20,10); textcolor(15);
    printf("Ur score is %d outof 30", marks);
      sleep(3);
      exit(1);
  }
  else
  {
     clrscr();
     sleep(1);
       gotoxy(20,8); textcolour(15);
           printf("Try next Time");
           sleep(2);
           ext(1);

    }
 getch();
}
```

Scanned by CamScanner

→ **string handling functions:**

- The functions which are used to handle the strings
- The string handling functions are from string.h headerfile

→ **strlen:**

It is always used to ~~writt~~ return length of the specified string along with spaces

syn: <variable> = strlen (str);

str → char[].

eg: char s[26] = "welcome";

i = strlen(s);

printf ("\n length is %d", i);

→ **strupr:**

It will return the string in upper case letters and stores them into the same variable

syn: strup(str);

eg: strupr(s); WELCOME

→ **strlwr:**

It will return the string in lower case letters and stores them into the same variable.

syn: strlwr (str);

eg: strlwr(s); welcome

→ **strrev:**

It will return the string into reverse and stores them into same variable

syn: strrev (str);

eg: strrev (s); emoclew

Scanned by CamScanner

→ **strcpy:**

It will copy the string from one to another.

syn: strcpy (str2, str1);

Eg: strcpy (f,s);

→ **strcat:**

It is used to concatinate two strings and stores them into the initial variable.

```
char f[10]= "sai",s[10]="baba";
```

syn: strcat (str1,str2);

Eg: strcat(s,f);
s= babasai
strcat(f,s);
f= saibaba

→ **NOTE:**

You can copy the value of one character variable to another character variable directly by using an operator '=' but it is not possible to copy the value of one character array to another character array using an operator equal but we make it possible by using strcpy();

```
char x='A',y;
    y=x; //valid

char x[10]="sai",y[10];
    y=x ;// invalid

    strcpy(y,x);
```

Scanned by CamScanner

→ **strcmp():**

It will compare two strings, if both strings are equal it will return 0. If the string1 is greater than string2 it will return positive value. If the string1 is less than string 2, it will return negative value.

The comparison is based on ASCII characters

syn : <variables> = strcmp(str1, str2);

i=strcmp(f,s)

→ **Program 158:**

Accept a string from the user, find out length of the string.

```
#include <string.h>
main()
{
 char s[20];
 int i;
 clrscr();
     puts("enter any string");
     gets(s);
     i=strlen(s);
     printf("\n length is : %d",i);
   getch();
}
```

→ **Program 159:**

Accept a string from the user, convert the string into upper, lowercase and ~~sw~~ vers reverse,

```
#include<string.h>
main()
{
char s[20];
int i;
```

Scanned by CamScanner

```
clrscr();
puts("Enter the string");
gets(s);
s[i]=strupr(s);
printf("\n length uppercase string : %s", s[i]);
printf("\n Ucase : %s", strupr(s));
printf("\n Lcase : %s", strlwr(s));
printf("\n rev : %s", strrev(s));
getch();
}
```

→ Program 160:

```
#include<string.h>
main()
{
char s[10]="sai";
clrscr();
printf("%s\t %s \t %s", strrev(s), strlwr(s), strupr(s));
getch();
}
```

output, ias ias ias

→ Program 161:

Accept a string from the user, and find out the given string is palindrome or not

```
#include<string.h>
main()
{
char s[10], f[10];
clrscr();
puts("Enter any string");
gets(f);
```

Scanned by CamScanner

```
strcpy (s,f);
 if (!strcmp(f , strrev(s)) )
  printf("\n Palindrome");
 else
    printf("\n Not ");
 getch();
}
```

Scanned by CamScanner

Functions

→ It is a self-contained block having a statement or group of statements to perform a particular task

1. With the help of functions, we can avoid repetition of code
2. Code optimization.
3. Memory optimization
4. Efficiency of the program will be increased.
5. Debugging the program becomes easier.

- Debugging is the process of executing the program in our own style by placing some temporary breaks inorder to identify the mistakes in the code.

*** "Function means which returns a value."

Functions are classified into 2 types:

1. Pre defined functions 2. User defined functions.

1. The functions which are provided by the c-compiler Another name for pre-defined functions are built-in functions
2. User defined functions are nothing but functions which are defined by us for our programming requirements It is also known as UDF or customized functions.

→ A function can be defined either above the main or below the main

→ Once we define the function, that function can be called 'n' no.of times either in the same program or in the different programs

→ We can call a function in another functions or we can call the same function to itself called recurssion functions

Scanned by CamScanner

→ Every predefined function is a user defined function

→ By the time of calling the function, that function should end with semicolon.

```
myfun()
{
}
main()
{
myfun(); // calling
}
```

→ Function defination

→ By the time of function defination, should not end with semicolon.

→ "main() is not completely predefined function and main is not completely uservdefined function."

→ It is a user defined function with pre-defined template (fixed name).

→ "A C program is a collection of both pre-defined and user defined functions. Thus C lang is called function oriented programming language (or) processor oriented programming language

→ Program 162:

```
main() // calling function
{
clrscr();
stars();
printf("\n Welcome\n");
```

Scanned by CamScanner

```
stars();
printf("\n to \n");
stars();
getch();
}
```

stars() called function

```
{
 int i;
 for (i=1; i<=10; i++)
  {
   printf("*");
   }
}
```

Function Prototypes:

These are classified into 4 types based on the arguments list and their return values

1) Function without argument without return value.
2) Function without argument with return value.
3) Function with argument without return value
4) Function with argument with return value

→ Syntax: [return type] <function name> ([list of args])

```
{
 [local variables];
  statements(s);
 [return <variable/value/exp>];
}
```

Scanned by CamScanner

→ Program 163:

Define a function to find the sum of two numbers.

```
main( )   /* Calling Function */
{
 int a,b;
 clrscr();
   printf("\n enter 2 numbers");
   scanf("%d %d", &a,&b);
  sum(a,b);
 getch();
}

 sum(int x,int y).  /* called function */

 {
   int s;
   s= x+y;
   printf("\n sum is : %d", s);
 }
```

→ Formal parameters Vs actual arguments:

- Formal parameters are the parameters which are declared in the function defination
- Formal parameters must be variables with corresponding data types
- Formal parameters are copies of actual arguments
- Formal parameters are acting as a local variables
- Actual arguments are the arguments which we are passing by the time of invoking the function.
- Actual arguments may be a values, variables or expressions.
- The type of both Formal and actual arguments must be same, but they are not in a same memory location.

Scanned by CamScanner

NOTE:-

Formal parameters must be variables with corresponding data type but it is optional in C language only for integers.

→ Program 164:

Define a function to find out the square of the given number

```
main() /* calling function */
{
  int n, r;
  clrscr();
    printf("\n enter a number");
    scanf("%d", &n);
  r = sq(n);
  printf("\n SQ is : %d", r);
  getch();
}
sq (int x) /* called function */
{
  int s;
  s = x * x;
  return s;
}
```

NOTE:

Whenever you want to use the value calculated by the called function into the calling function then we need to define a function with return.

Scanned by CamScanner

→ **Program 165:**

Define a function to find out average of 3 numbers.

```
float favg (int x, int y, int z); /* Function Declaration */
~~a= favg (m1, m2, m3)~~
main( )
{
 int m1, m2, m3;
 float a;
 clrscr();
    printf (" In Enter 3 sub marks : ");
    scanf (" %d %d %d ", &m1, &m2, &m3);
 a=favg (m1, m2, m3);
 printf ("In Avg is : %2f ", a);
 getch();
 }
 float favg (int x, int y, int z)
 {
   float a;
   a= (x+y+z)/3;
   return a;
 }
```

NOTE:

If we define any function with return type the corresponding function declaration must be written either in the main or above the main, it is optional in C-compiler if the function return type is an integer.

Scanned by CamScanner

→ Program 166:

Define a function to find out factorial value of given no.

```
main()
{
 int i,n;
 clrscr();
 printf ("\n Enter a number");
 scanf ("%d", &n);
 i=fact(n);
 printf ("\n Fact is :%d ",i);
 getch();
}
 int fact (int x)
{
   int i, f=1;
   for (i= x; i >=1; i--)
    { f= f* i; }
   return f;
}
```

→ Program 167:

Define a function to find out the given number is odd or even

```
void oddorEven (int x)
{ (x%2=1)?printf("\n ODD"): printf("\n Even");}
main()
{
 int n;
 clrscr();
```

Scanned by CamScanner

```
printf("\n enter a number");
scanf ("%d",&n);
oddorEven(n);
getch();
}
```

NOTE:

** → If we define any function with return type then the corresponding declaration must be re written either in the main or above the main but it can be eliminated by defining the function above the main

→ when a called function is not returning any value to the calling function then we need to write void as a returntype.

→ void is an empty data type (or) 0 byte data type.

→ Program 163.

Q) How can you find number of bytes taken by a variable or a data type or a function by using sizeof?

```
void main( )
{
  int x;
  clrscr();
  printf("\n size of x is %d", sizeof(x));
  printf("\n size of float is %d", size of (float));
  print ("\n size of i's : %d", sizeof (clrscr()));
  getch();
}
```

Scanned by CamScanner

→ Programs 169:

Q) What is the size of (sizeof)? sizeof (sizeof)?

compilation error

But it is a keyword and operator

Q) When to write void as a return type for the main function? What is the difference between void main and int main?

The main function always used to return int type bytes to operating system implicitly. Hence we need to write integer int to return type for main function

Similarly if you want to kill the nature of returning int types bytes to OS, we need to write void as return type.

Void:

Void is empty or zero type data type

Function with Arrays

→ It is the process of defining the function by passing an array element or by passing an entire array to the function as a function argument.

→ Program 169:

Define a function to print even elements for the array

```
void evenElement (int x)
{
    if (x%2==0)
        printf ("\n%d", x);
}
void main ( )    [ int main() }
{
  int a[3], i;
  clrscr();
```

Scanned by CamScanner

```
for (i=0; i<3;i++)
 {
   printf ("\n enter number");
   scanf ("%d", &a[i]);
   }
  for (i =0, i<3;i++)
  { evenElement (a[i]);}

getch();
}
```

⇒ <u>Program 170:</u>

Define a function to find out sum of array elements

```
int sumAry (int x[])
 {
  int s=0,
   for (i=0, i<3;i++)
    { s = s+x[i];}
   return s;
 }
 int i; /* global variable*/
 main()
 {
  int a[3]; /* local variable */
   clrscr();
   for (i=0;i<3;i++)
    {
     printf("\n enter number");
      scanf ("%d", &a[i]);
    }
  f= sumAry(a);
```

Scanned by CamScanner

```
printf ("In Sumary %d",1);
getch();
}
```

→ Local variable vs Global variable:

- A variable which is declared within a function, the variable can be used in the function only
- The default value for the local variable is garbage value
- A variable which is declared above the main, the variable can be used into the any of the function used in the program known as global variable
- The default value of global variable is zero

→ NOTE:

As per the programming requirements we can declare both local and global variables with the same name, At that time the compiler will give priority for the local variable

→ Program 171:-

```
int x=10;
void main()
{
int x=20;
printf("%d",x);
}
printf ("%d",x);
```

compilation error:

This is written out of main. so error.

→ Program 172:

```
int i=99;
main()
{
int i=200;
printf ("%d",i);
}
```

output: 200

Scanned by CamScanner

→ Program 173:

```
int i=90;
main()
{ int i=99;
    printf ("%d", i);
    myfun();
}
myfun()
{ printf ("%d", i); }
```

output: 99 90

→ Scope of variables

- A variable which is declared within a scope, the variable can be used in the scope only.
- If we declare any variable in outer scope, the variable can be used in an outer scope as well as inner scope
- We should not declare more than one variable with a same name in a scope

→ Program 174:

```
main()
{
 int x=111;
  {
   int y=222;
   printf ("\n Inner y: %d", y);
   printf ("\n Inner x: %d", x);
  }
 printf ("\n outer x: %d", x);
 /* printf ("\n outer y: %d", y); CE */
 getch();
}
```

Scanned by CamScanner

→ **Program 175:**

```
main()
{
 int x=10;
  {
   int x=20;
     {
      int x=30;
      print("%d",x);
     }
   printf("%d",x);
  }
 printf("%d",x);
}
```

→ **Function with Strings**

- It is process of defining a function with character array.

→ **Program 176:**

Define a function to print the given string in vertical.

```
int i;
void stver (char x[])
{
    for ( ;x[i]!='\0';i++)
     { printf ("\n%c", x[i]);}
 }
void main()
{
 char s[20];
 clrscr();
   puts("Enter any string");
   gets(s);
```

Scanned by CamScanner

```
strver(s);
getch();
}
```

→ program 177:

To find out length of a given string without spaces

```
int i, len;
int strlenws (char x[])
{
  for (; x[i]!='\0'; i++)
     { if (x[i] =32)
         len++;
     }
   return len;
}

void main() {
 char s[20];
 clrscr();
 puts ("Enter any string");
 gets(s);
 i = strlenswos(s);
 printf ("\n length without spaces : %d ", i);
 getch();
}
```

Scanned by CamScanner

Program 128:

Togal case

```
int i;
void strtgl (char x[] )
{
  for ( ;x[i]!='\0' ;i++)
    {
      if (x[i] >=65 && x[i]<=90)
        x[i]+=32;
        else if(x[i]>=97 && x[i]<=122)
          x[i]-=32;
      }
    printf ("\n string in tglcase : %s", x);
}
void main( )
{
  char s[20];
  clrscr();
   puts ("Enter any string");
   gets(s);
   strtgl (s);
   getch();
}
```

Output:

Enter any string welcome to SSSIT

WELCOME TO sssit

Scanned by CamScanner

COMMAND LINE ARGUMENTS

Q) Can you arguments to main function?

Yes, it is possible through command line arguments.

- command line arguments is a process of passing the values to program by the time of execution along with program name through command line
- Main function can take two arguments int

main (int argc, char * argv[])

- argc is an integer variable, it can hold number of arguments we passed to the main function through command line.

→ argv[] is a character array, it can hold the values which we are passing through command line.

→ argv[0] can hold a file name from argv[1] to argv[n] is holding the values

→ we can pass 'n' no. of values, any type of values they should be seperated by space and they are treated by the system as the strings only.

- In order to pass the values, we need to select argument from options menu.

→ Program 179:

```
void main (int argc, char * argv[])
{
  int i;
  clrscr();
  printf("\nNo. of. CMD args %d ", argc);
  printf("\nName of Prg File : %s", argv[0]);
  for (i=1; <argc ; i++)
```

Scanned by CamScanner

```
{ printf ("\n %s", argv [i]);}
getch();
}
```

→ **Program 180:**

Program to find sum of two numbers by taking the values through command line.

```
void main (int argc, char * argv[])
{
  int x,y,s;
  clrscr();
    x = atoi (argv [1]);
    y = atoi (argv [2]);
    s = x+y;
    printf ("\n sum is : %d", s);
  getch();
}
```

→ **NOTE:**

In order to convert string type to integer type, string type to float type, string type to long type we need to make use of following macros

atoi() string to int
atof() string to float
atol() string to long

These are from stdlib.h header file.

Scanned by CamScanner

→ Program 181:

Define a function to find out sum of command line arguments

~~#inclu~~

NOTE:

argc argv [] are the formal parameters to the main function, hence they are acting as local variables to the main function only. But we have two global variables (_argc _argv[]) - are the global variables from dos.h header file. It will work exactly similar with argc and argv[] array. But these ~~can~~ be are works to the local to main function, but _argc and _argv[] will be used for any function

```
#include <dos.h>
void main (int argc, char* argv[])
{
 clrscr();
 sum();
 getch();
}
sum()
{ int i, s=0;
  for (i=1; i< _argc; i++)
   { s=s+atoi(_argv[i]); }
  printf("\n sum is %d", s);
}
```

Scanned by CamScanner

Pre processor directives

→ Pre processor bdirectives are prosscessed before actual process gets started.

→ with the help of Pre. Processor Directives, we can reduce the code (code optimization), memory optimization, efficiency of the program increases.

→ we can write the Pre processor directives anywhere in the program but it must begin with operator pound (#) and it should not end with semicolon (;):

→ These are classified into 3 types.

(i) Macro substitution (# define)

(ii) File inclusion (#include)

(iii) Conditional control substitution macros (#if, #else; #endif, #undef, # ifndef, #ifdef)

→ Macro substitution:

Syn: #define <macro> <macrodef>

- Another name for macro's is symbolic constants. The macro definition will be substituted by the complier automatically by the time of execution wherever we use macros.
- Macro defination can be a value or an expression or a function
- with the help of Macro's we can elimate to define ~~syn~~ simplified functions
- Simplified function is a function which is having very less number of statements

Scanned by CamScanner

→ Eg:

```
#define printf pf
# define scanf sf
#define wait getch()
void main()
{
int a,b;
clrscr();
pf ("\n Enter 2 numbers");
sf ("%d %d", &a, &b);
printf("\n Sum is : %d", a+b);
wait;
}
```

→ Program 182:

```
#define MAX 10
main()
{
int a[MAX] ,i;
clrscr();
for (i=0; i< MAX ; i++)
  {
    Printf ("\n Enter an element ");
    scanf ("%d ", &a[i]);
  }
for (i=0, i< MAX ; i++)
  {
    printf ("\n %d ", a[i]);
  }
getch();
```

Scanned by CamScanner

→ NOTE:

As per the programing requirements we can pass the parameters to the macros, if necessary we can use a macro inside a macro

→ Program 183:

```
/* int sq(int x)
 {
  int s;
  s=x*x;
  return s;
  }          */

/* int sq(int x)
 { return (x*x); } */

#define sq(x) (x*x)
#define cu(x) (sq(x)*x)
#define MAX(x,y)  (x>y)?x:y
void main()
 {
  int n;
  clrscr();
    printf("\n Enter a number");
    scanf("%d",&n);
    printf("\n SQ is: %d", sq(n));
    printf("\n CU is: %d", cu(n));
    printf("\n Biggest is: %d", MAX(120,20));
   getch();
 }
```

Scanned by CamScanner

→ File inclusion:

whenever you defined a function in the program that function can be used for 'n' no. of times in the same program but it cannot be accessed or used in other program but we make it possible using file inclusion.

i.e.,

(i) whenever you require any function for multiple number of times either in same program or another program then it is recommended to define the corresponding functions in a seperate file and save the file with a seperate name

(ii) whenever you require any of the function in the program then it is recommended to use the corresponding file-name as and file inclusion

step 1: select new from File Menu

step 2: define all required functions

Eg:
```
void blink(char x[3])
{ int i;
  for(i=1 ; i<t++)
   {
    delay(100000);
    textcolor(1);
    gotoxy(29,10);
    printf("%s", x);
   }
}
```

step 3: select save from the File Menu and save file with Name: Mylib123.c

Scanned by CamScanner

Step 4: Select New from the File Menu

Step 5: Type the following code and Run.

```
#include "mylib123.c"
void main()
{
  char s[20];
  clrscr();
      puts("Enter any string");
      gets(s);
      blink(s);
      getch();
}
```

→ conditional control substitution macros:-

#ifdef: It returns true if the specified macro is already defined

#ifndef: It will return true if the specified macro is not defined.

#undef: It is used for undefining the macro which is already defined

→ Program 184:

```
#define MAX 10
void main()
{
  clrscr();
  #ifdef MAX
     printf("\n MAX = %d", MAX);
     #endif
     #ifndef MIN
     #define MIN 5
     #endif
```

Scanned by CamScanner

```
#ifdef MIN
printf ("\n MIN = %d", MIN);
#undef MIN
#endif
/* printf ("\n MIN = %d", MIN); CE */
getch();
}
```

Scanned by CamScanner

Pointers

- It is a variable always used to hold address of the variable.
- With the help of pointers we can establish the link b/w the variables
- It will increase the efficiency of the program as well as it will extend the memory dynamically.
- Syntax: <datatype> * <vname>;

  Eg: int * ptrx;
  float * ptry;
  char * ptrz;
- * ptrx is an integer pointer variable. It can hold the address of any integer variable
- * ptry is a float pointer variable. It can hold the address of any float variable
- * ptrz is char pointer which can hold the address of any char variable.
- Every pointer variable will take 2 bytes of space in the memory irrespective of its datatype.

→ Program 185:

```
main()
{
int * ptrx;
float * ptry;
char * ptrz;
clrscr();
printf("\n size of is:%d ", sizeof (ptrx));
```

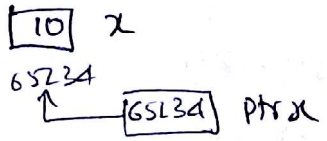
Scanned by CamScanner

```
printf ("\n size is : %d", sizeof (ptry));
printf ("\n size is : %d", size of (ptrz));
getch();
}
```

→ Eg: int a; *ptr x;

a = 10;

ptr x = &x;



→ Program 186:

```
main()
{
 int x, *ptrx;
 clrscr();
  x = 100;
  ptrx = &n;
  printf ("\n ADD of x : %u", &x);
  printf ("\n ADD of x : %u", ptrx);
  printf ("\n value of x : %d", *ptrx);
  printf ("\n value of x : %d", *&x);
  printf ("\n value of x : %d", *(&x));
  printf ("\n value of x : %d", *(n));
  printf ("\n value of x is %d", x);
 getch();
}
```

*→ *ptrx is nothing but we are getting the value of the variable. the point *ptrx is pointed to

Scanned by CamScanner

→ program 187:

```
main()
{
int x,*ptrx,y,*ptry;
clrscr();
x=10; y=5;
ptrx=&x ; ptry=&y;
*ptrx +=* ptry;
printf ("\n x=%d  y=%d", *ptrx,*ptry);
*ptrx-= *ptry;
printf ("\n x=%d  y=%d", *ptrx,*ptry);
*ptrx= *ptry;
printf ("\n x=%d  y=%d", *ptrx ,*ptry);
getch();
}
printf ("\n%d", *ptrx+y);
printf ("\n%d", x-*ptry);
printf ("\n%d", *ptrx**ptry);
printf ("\n%d", *ptrx / (*ptry));
```

Scanned by CamScanner

Types of Pointers

(i) **Wild pointer:**

It is a pointer variable which is not initialized with any address. It is also known as bad pointer

Eg:- main()

```
{
  int *ptr; /* wild pointer */
  clrscr();
  printf("%d", *ptr);
  getch();
}
```

(ii) **Null pointer:**

Null pointer is a pointer variable which is initialised with null

Eg: int *ptr = NULL; or (int *)0;
float *ptr = NULL; or (float *)0;
char *ptr = NULL; or (char *)0;

→ **Program 188:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int * ptr=NULL;
  clrscr();
  if (ptr == (int*)0)
  printf("\n Both are same");
  else
  printf("\n Both are not same");
  getch();
}
```

o/p Both are same

Scanned by CamScanner

***NOTE:

We should not call a function whose return type is void as a part of printf. In all such cases, the compiler will raise an error.

Eg: Program 189

```
void oe(int x)
{
  if (x%2==0)
  printf ("\n Even");
  else
  printf ("\n odd");
}
void main()
{
  clrscr();
  printf (oe (10)); //error
  getch();
}
```

Void pointer:

Integer pointer variable can hold the address of any integer variable, similarly a float pointer variable can hold the address of any float variable whereas void pointer is a generic pointer. It can hold the address of any variable of any type.

While retrieving the data from the variable using void pointer, we require to perform corresponding type casting to the corresponding types.

Eg: Program 190:

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

Scanned by CamScanner

```
float y= 3.114.f;
char z='A';
void *ptr;
clrscr();
ptr=&x;
printf("\n x=%u", *(int *) ptr);
Ptr=NUL;
```

Pointer Arithematics:

With the help of pointer arithematics, we can find the next location or previous locations of the variable.

- If we use increment operator on a ordinary variable, then value of the variable will be increased by 1. Similarly if we use decrement operator on the ordinary variable then value of the variable will be decreased by 1.
- If we use increment operator on the pointer variable, it will move to the next location.
- If we use decrement operator on the pointer variable, it will move to its previous location.
- The movement is completely depended upon number of bytes taken by the pointer variable data type

Eg:

```
void main()
{
 int x,* ptr;
 clrscr();
 ptr=&x;
 printf("\n Add of : %u", ptr);
 ptr++;
 printf("\n Next loc of x is: %u", ptr);
```

Scanned by CamScanner

```
ptr--;
printf("\n Pre loc of x is: %u", ptr);
}
```

o/p Add : 65424

Nex loc of x is : 65426

Pevloc of x is : 65424

→ <u>Pointer with Arrays:</u>

It is the process of accessing the array elements using address of each index rather than using index

```
main()
{
  int a[5] = {10,20,30,40,50}, i, *ptr;
  clrscr();
  /* ptr = &a[0]; or ptr = &a; */
  ptr = a; /* Base connection of an array */
  for (i=1; i<=5; i++)
  {
    printf("\n %d", *ptr);
    ptr++;
    sleep(1);
  }
  ptr = &a[4];
  for (i=1; i<=5; i++)
  {
    printf("\n %d", *ptr);
    ptr--;
    sleep(1);
  }
  getch();
}
```

Scanned by CamScanner

→ NOTE:-

In order to access the array elements doesnot require to have any explicit pointer variable just because the array name itself it is acting as a pointer variable.

Program 192:

```
main()
{
int
int a[] = {10,20,30};
printf("%u", &a[0]); & printf("%u", (a+0));
printf("%u", &a[1]); & printf("%u", (a+1));
printf("%d", *&a[2]); & printf("%d", *(a+2));
```

→ Program 192:

```
main()
{
int x[] = {10,20,30,40,50};i;
clrscr();
for(i = 0; i<5; i++)
  {
    sleep(1);
    printf("\n Add ress of at x[%d] = %u", i, (x+i));
    printf("its value is: %d", *(x+i));
  }
getch();
}
```

Scanned by CamScanner

→ Strings with pointers

193: Program to print given string int in the vertical.

```
main()
{ int T[4]; }

printf.

main()

{ char s[10]="welcome";

int t;

clrscr();
printf ("\n %c", s[0].);
printf ("\n %c", *(s+0));
printf ("\n %c", *(0+s));
printf ("\n %y14 %c :", -
getch();
}
```

→ p94: strupr();
main();

```
void {
main clrs( raybe)
```

Scanned by CamScanner

195: Accept of string ~~the~~ ~~fol~~ ~~lowt~~ print the given string in the given position.

```
main()
{
 char s[20];
 int i,j;
 clrscr();
  puts ("Enter any string");
  gets(s);
   for (i=0; *(i+s)!='\0'; i++)
    {
     for(j=1; j<=(i+1); j++)
      {
       printf ("%c", *(i+s);
      }
     printf ("\n");
    }
  getch();
}
```

→ Pointer to Pointer:

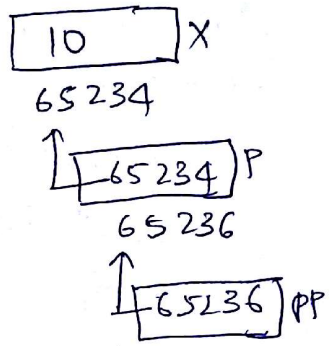
- A variable is a space to store the data.
- Every variable is having some adress in the memory to hold the adress of the variable we require a pointer variable
- Pointer variable is also a variable it is also having some adress in the memory.
- In order to hold the adress of the pointer variable we require another pointer variable called pointer to pointer.

Scanned by CamScanner

- It will be the efficient upto 24 pointers.

representation :

```
int x, *p, **pp;

x=10;

p=&x;

pp=&p;

printf ("\n %u", p); &x

printf ("\n %d", *p); 10

printf ("\n %u", pp); &p

printf ("\n %u", *pp); val of p
                       i.e. &x

printf ("\n %d", **pp); x val 10
```



→ Function with pointers:

These are ~~dalsified~~ classified into two types

(i) call by values

(ii) call by reference

- call by values is a process of calling a function by passing values of the variable to the function as a function argument.
- In the call by values the values are reflected in the formal parameters
- Call be reference is a process of calling a function by passing reference of the variable (adress of the variable) to the ~~f~~unction as a function argument.
- In the call by reference the values are reflected in the actual arguments.

Scanned by CamScanner

Program 196:

```
main()
{
 int a,b;
 clrscr();
 pf ("\n Enter 2nos");
 sf ("%d %d", &a,&b);
 swap (a,b);
 getch();
}
swap (int x, int y)
{
 int temp;
 temp=x;
 x=y;
 y= temp;
 pf ("\n AfC %d %d ",x,y);}
}
```

```
main()
{
 int a,b;
 clrscr();
 pf ("\n Enter 2no's"),
 sf ("%d %d", &a, &b);
 swap(&a, &b);
 getch();
}
swap (int *x, int *y)
{
 int temp;
 temp =*x;
 *x=*y;
 *y = temp;
 pf ("\n AfC %d %d", *x, *y);
}
```

Scanned by CamScanner

Structures

→ struct is a key word

→ As per the programing requirements, whenever you are not satisfied with pre defined datatypes we can define our own data type called user defined datatype

→ In the c-lang we have the following keywords like union, struct, enum and typedef

→ struct is a key word, with the help of struct keyword we can define our own data type called user defined data type

→ structure allows you to store heterogenous mixture of elements to maintain and manipulate records

→ we can define the structure either above the main or within the main.

→ Structure is a logical representation, it doesnot ocupy any space in a memory, hence it doesnot allow you to store the data

→ In order to store the data we need to create the physical representation for a structure called structure variable.

→ Based on the programing requirements we can define 'n' no. of variables for the structure

→ The size of the structure variable is completely depends upon total size of all the mmembers of the structure

→ whenever we create a variable for the structure, ultimately whatever the members are existed in the structure, for those members memory is allocated within the structure variable

→ To access the members of the structure we need to make use of member access operator(.) period.

Scanned by CamScanner

Syn: <datatypes <vnames>;

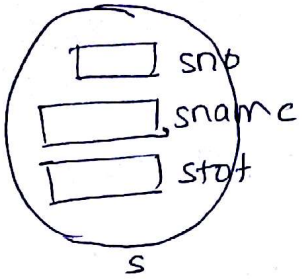
Eg:- int n;

struct student s;

s.sno=101;

s.sname="priya";

s.stot=999;



Syn: <struct> <structnames>

{

elements of member

} [variable];

Eg: struct student {

int sno;

charsname[20],

int stot;

};

→ Program 197:

```
main()
Struct Myvalues /* Myvalues UDdatatype*/
{
    int x=100;
        y=200;
  };
Struct Myvalues m;
clrscr();
printf("\n x value is : %d", m.x);
printf ("\n y value is : %d", m.y);
getch();
}
```

Error: Declaration missing ; in the function main

NOTE: structure numbers cannot be initialised by the time of defining the structure but it is always recommended to assign the values to the members of the structure after creating the struct variable

→ **Program 198:** Example for static initialisation

```
main()
{
 struct myvalues
 {
   int x;
   float y;
  char z;
 };
struct MyValues m = {10, 3.14f, 'A'};

 clrscr();
 printf ("\n size of m is:%d ", sizeof (m));
 printf ("\n x value is: %d", m.x);
 Printf ("\n y value is: %.2f ", m.y);
 printf ("\n z value is : %c ", m.z);

 getch();
 }
```

→ **Program 199:**

Program to store a student details with s.number, s.name and s.total marks.

```
main()
{
 struct student
 {
    int sno;
   char sname[20];
   int stot;
 };
```

Scanned by CamScanner

```
main()
{
  struct student s;
  clrscr();
  printf("\n Enter sno, sname, stot");
  scanf(" %d %s %d", &s.sno, &s.sname, &s.stot);
  printf(" \n stud sno is : %d", s.sno);
  printf(" \n sname is : %s", s.sname);
  printf("\n stot is : %d", s.stot);
  getch();
}
```

Structure with Arrays:

→ Program 200:

Program to store 3 employees information with ename, salary, bonus and net salary. If salary is greater than 10,000 bonus is 1,000 else bonus is 500

```
#define MAX 3
struct employee
{
  char ename[20];
  int sal, bo, net;
}e[MAX];
int i;
void main()
{
  /* struct Employee e[MAX]; */
  clrscr();
```

Scanned by CamScanner

```
for (; i<MAX; i++)
{
  Printf ("\n Enter Ename and salary");
  scanf ("%s%d", &e[i].ename, &e[i].sal);

  if (e[i].sal>10,000)
    e[i].bo=1000;
  else
    e[i].bo= 500;
  e[i].net = e[i].sal + e[i].bo;
}
clrscr();

for (i=0; i<MAX; i++)
{
  printf("\n ==========");
  printf("\n sname is : %s", e[i].ename);
  printf ("\n sal is : %d", e[i].sal);
  printf ("\n Bon is : %d", e[i].bo);
  printf ("\n Net is : %d", e[i].net);
  printf ("\n ----------");
  sleep(1);
}
getch();
}
```

Scanned by CamScanner

→ Program 201:

Program to store 3 pre-book details with book name, prize, quantity and bill.

```
#define MAX 3
struct book
{
  char bname[20];
  int p,q,bi;
} b[MAX];

int i;
void main()
{
  clrscr();
  for (; MAX>i; i++)
  {
    printf("\n Enter Book %d details ", (i+1));
    printf("\n Enter Bname, Price and Qua");
    scanf("%s %d %d", &b[i].bname, &b[i].p, &b[i].q);
    b[i].bi = b[i].p*b[i].q;
  }
  clrscr();
  printf("\n ==========");  printf("\n Bname \t Price \t Quant \t Bill");
  for (i=0; i< MAX; i++)  printf("\n ----------");
  { printf("\n %s\t %d\t %d\t %d", b[i].bname, b[i].p, b[i].q, b[i].bi);

    sleep(1);
  }
  printf("\n ==========");
```

Scanned by CamScanner

Function with structures:

This is a process of defining the function with user defined data types (complex datatypes)

→ Program 202:

Define a function to find out sum of the mambers of the structure

```
struct MyValues
{
  int x;
  int y;
}m;

int sum (struct MyValues d)

{ int s;
  s= d.x+d.y; return(s); }

void main()
{
  clrscr();
  printf ("\n Enter 2 numbers");
  scanf ("%d %d", &m.x, &m.y);
  printf ("\n sum is = %d", sum(m));

  getch();
}
```

Scanned by CamScanner

→ **Program 203:**

Define a function to compare members of 2 complex variables.

```
struct {My Values
{
  int x;
  int y;
} m1, m2;
void compare (struct My values d1,
              struct My Values d2)
{
    if (d1.x == d2.x && d1.y == d2.y)
      printf ("\n Both are same");
    else
      printf("\n Both are not same");
}
void main()
{
  clrscr();
    printf("\n Enter two values");
    scanf ("%d %d", &m1.x, &m1.y);
    printf ("\n Enter two values");
    scanf ("%d %d", &m2.x, &m2.y);
    compare (m1, m2);
  getch();
}
```

Scanned by CamScanner

Structure with Pointers:

→ Structure pointer variable always holding the adress of the structure and it is pointing to the initial member of of the structure.

→ To access the members of the structure we need to make use of -> as a member access operator

→ By using structures with pointers we can access the members of the structure through pointers

→ Program204:-

```
Struct MyValues
{
  int x;
  int y;
} m, *ptr;

void main()
{ clrscr();
   printf ("\n Enter 2 numbers");
   scanf ("%d %d", &m.x, &m.y);

   ptr = &m;

      printf ("\n x val : %d", *ptr);
      printf ("\n x val : %d", ptr -> x);
      printf ("\n y val : %d", ptr -> y);

   getch();
}
```

Scanned by CamScanner

Nested Structures

→ It is a process of defining a structure not only with pre-defined datatypes if required we can also use user defined datatype as a member of a structure

→ Program 205.

```
struct Doj
{
  int dd, mm, yy;
};

struct Employee
{ int eno;
  char ename[20];
struct doj d;
};e;
void main()
{ clrscr();
  Printf("\n Enter eno, ename");
  scanf("%d%s", &e.eno, &e.ename);
  printf("\n Enter Doj");
  Printf("\n Enter DD, Month, Year");
  scanf("%d%d%d", &e.d.dd, &e.d.mm, &e.d.yy);
  clrscr();
  Printf("\n ENO is : %d", e.eno);
  printf("\n Ename is : %s", e.ename);
  printf("\n DOJ is : %d-%d-%d", e.d.dd, e.d.mm, e.d.yy);
getch();
}
```

Scanned by CamScanner

Union

→ Union is a key word, with the help of union key word we can define our own data type called user defined data type

→ The union will work exactly similar with structure except sharing the memory

→ The size of the structure variable is completely depends upon the total size of all the members of the structure. Hence individual memory blocks are created for each element of structure.

→ The size of union variable is completely depends upon largest element in the union members and this memory can be shared by each member of union. Thus writing new data will over write old data. Thus there is a chance of loss of data

→ Accessing the members of the structure as same manner as accessing the member of the union

```
<struct><structnames>
{ element(s);
} [variables];
void main() {
struct Myvalues {
int x; float y;
} m;
m.x = 10; m.y = 3.1f;
pf("\n x is: %d", m.x);
pf("\n y is: %f", m.y);
getch();
}
```

x y m
10 | 3.14
2bytes 4bytes
6 bytes

```
<union><union name>
{ element(s);
} [variables];
void main() {
union Myvalues {
int x; float y;
} m;
m.x = 10; m.y = 3.14f;
pf("\n x is: %d", m.x);
pf("\n y is: %f", m.y);
getch();
}
```

3.14f
10 x/y
4bytes.

6B-U

3.14f

Scanned by CamScanner

→ **Enum** (enumeration datatype):

- With the help of enum datatype we can define our own datatype called userdefined datatype.
- In general, in order to make any variable value as a constant we need to make use of the key word constant.

```
int x=10;
    x=x+10;
    printf ("\n %d", x);   output: 20
    const int x=100;
    printf ("\n %d", x);   output: 100
        x=x+0; // compile type error: The constant object
                  cannot be modified.
```

→ With the help of enum keyword we can create, list of integer constants only

- The integer constant values are starts from 0 to (n-1).
- If required we can also configue the desired constant values. But the series will be effected by one from the point where we change the constant.

Syntax: enum<identifier>{const1, const2, ---, const n};

Program 206:

```
# include <stdio.h>
# include <conio.h>
main()
{
    enum HNAMES{ JAN=1, FEB, MAR, APR, MAY, JUN, JULY, AUG, SEP, OCT,
                 NOV, DEC}
    const int x=10;
    int i;
    clrscr();
    printf("\n x = %d", x);
    printf("\n JAN = %d", JAN);
```

Scanned by CamScanner

```
printf("\n DEC= %d", DEC);
for(i=JAN; i<=DEC;i++)
  { printf("\n %d", i); }
    text colour (YELLOW);
     (printf("HELLO");
     getch();
}
```

→ <u>Typedef</u>: it is the keyword

- With the help of this keyword we can assign our own names to the datatypes.

<u>syn</u>: typedef < type-defination > < identifier >;
↓ any datatype ↓ any name.

<u>Program</u> -

```
typedef struct
{ int sno;
  char sname [20];
  int stot;
 } student;
/* struct student
  { int sno;
   char sname [20];
   int stot; };
   struct student s;
   typedef struct student stu; */
void main()
{ typedef int integer;
   typedef float real;
      Integer x=10;
       real y= 3.14f;
       student s={101, "Priya", 989}; }
      clrscr();
```

```
printf("\n x val: %d", x);
printf("\n y val : %d", y);
getch();
```

Scanned by CamScanner

Files

- Usually the values which we are entering during program execution as an input as well as the values which we are getting after executing the program, these values will be stored in the system memory temporarily.
- In order to store the data permanently, we need to make use of persistency programs. It is possible by using.
  (1) Files
  (2) Data Bases.
- Files are used to store the data permanently in a particular space on the disk
- In order to work with the files in C-compiler we have to use the functions provided by stdio.h

→ File handling functions:

In order to interact with the files, compulsorily we required a file pointer in order to access the file internally in the memory

syn: FILE * <variable>;

Eg: FILE *fp;

- FILE is a structured datatype, hence it can store heterogenous mixture of data.
- FILE is a structed datatype in order to make the differentiation between regular datatype to structured type it is recommended to write in capital letter.

fopen():

It is a function used to open a file in the memory.

- fopen() for required two arguments "filename" and "filemode"

syn: <fp> = fopen ("Filename", "filemode");

Scanned by CamScanner

- In clanguage we have the following file modes

1) "W": It will open the file for the purpose of writing the data. If the specified file is already existed, it will open the file and it will overwrite old with new data.

2) "r": It will open the file for the purpose of reading. If the specified file is not existed in read mode, fopen function will return NULL.

3) "a": If will open the file for appending purpose i.e., if the file is already existed, it will open the file and add new data into it. If the file is not existed, it will create the file and allows for writing.

4) "w+": The main moto is for writing optional reading.

5) "r+": Main moto is reading optional writing.

fputc():

It is an output function which enable upon you to write the data into the file.

syn: fputc (<variable>, <fp>);

fgetc():

It is an input function. It will get the character from the file and assign to the specified variable.

syn: <variable> = fgetc(<fp>);

fclose():

It is used to close the specified file

syn: fclose (<fp>);

Scanned by CamScanner

→ Program 207:

Program to store a character into a file

```
#include <stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char x;
    clrscr();
    fp = fopen("sample", "w");
    printf("\n enter any char");
    scanf("%c", &x);
    fputc(x, fp);
    fclose(fp);
    printf("\n FILE IS CREATED");
    getch();
}
```

→ Program 208:

```
void main()
{ FILE *fp;
  char x;
  clrscr();
  fp= fopen("sample","r");
  if(fp==NULL)
  { printf("\n sorry file is not found"); }
  else
   { x= fgetc(fp);
     printf("\n %c", x); }
   fclose(fp);
  getch();
}
```

Scanned by CamScanner

→ **Program 209:**

Program to store data into the file

```
void main()
{
 FILE *fp;
 char x;
 clrscr();
 fp=fopen("Mydata","w");
printf("\n Enter ur data Press *
                for exit");
 do
 {
  scanf("%c",&x);
   if (x!='*')
    { fputc(x,fp); }
    else
    {break;}
  } while(1);
 fclose(fp);
 printf("\n File is saved");
 getch();
 }
```

→ usually every string ends with null character '\0', similarly every file is end with EOF

EOF — end of file

→ **Program 210:**

Program to read the data from the file

```
void main()
{
 FILE *fp;
 char x;
 fname[20];
 clrscr();
 puts("Enter any filename");
 gets(fname);
 fp=fopen(fname,"r");
 if (fp==NULL)
printf("\n sorry file not found");
else
{ while(1)
  {
   delay(100000000);
   x=fgetc(fp);
   if (x==EOF)
    break;
   else
    printf("%c",x);
  }
}
fclose(fp);
getch();
}
```

Scanned by CamScanner

→ Program 21:

Program to copy the data from one file to another file

```
#include <stdio.h>
#include <conio.h>
void main()
{
  FILE *fps, *fpd;
  char x;
  clrscr();
     fps=open ("student.c", "r");
     if (fps==NULL)
        printf("\n SORRY FILE IS NOT FOUND");
        steep(1);
     else
      {
       sleep(1);
       printf("\n FILE IS FOUND");
       sleep(1);
       printf ("\n FILE IS COPYING -----");
       fp d= fopen ("mystu", "w");
       while (1)
        {
         delay(10000);
         x = fgetc(fps);
          if (x==EOF)
           break;
          else
          { fputc(x,fpd); }
        }
```

Scanned by CamScanner

```
        sleep(1);
        printf("\n File is copied");
    }
    fclose(fps);
    fclose(fpd);
  getch();
  }
```

→ **Dealing with records:**

– **fprintf();**

It is an output function it is used to perform the formated output to the stream (file)

syn: fprintf(<fp>,"control strings', variables);

→ **fscanf():**

It is an input function used to perform the formatted input from the stream

syn: fscanf(<fp>,"control strings", &variables);

→ **Program 212:**

Program to store s students information.

```
void main()
{
FILE *fp[1];
int sno, tol,i;
char sname[20];
clrscr();
  fp=fopen("student","a");
```

Scanned by CamScanner

```
for (i=1; i<=s ; i++)
  {
   printf ("\n Enter sno, sname and stot"),
   scanf (" %d %s %d", &sno, &sname , &tot);
   fprintf (fp, "\n%d \t %s\t %d", sno, sname, stot);
  }
  printf ("\n File is created");
  fclose (fp);
  getch();
 }
```

→ Program 213:

```
#include <stdio.h>
#include<conio.h>
void main()
{
 int no, t;
 char name[20];
 FILE *fp;
 clrscr();
   fp= fopen ("student", "r");
    if (fp==NULL)
        printf ("\n File is not found");
    else
   {
   while ( fscanf (fp, "%d %s %d", &no, &name, &t)!=EOF)
   {
    Sleep(1);
    printf ("\n %d \t %s \t %d ", no, name, t);
   }
  }fclose (fp);
   getch();
```

Scanned by CamScanner

→ Program 214: Accept student no. from the user, display the corresponding student details if exist.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 FILE *fp;
 int no, t, sno, found = 0;
 char name [20];
 clrscr();
  fp=fopen ("student", "r");
   if(fp==NULL)
   printf ("\nFile is not found");
   else
   {
     sleep(1);
     printf ("\n Enter student number:");
     scanf ("%d", &sno);
     while (fscanf (fp, "%d%s%d", &no, &name, &t)!=EOF
       {
        if (sno==no)
        {
         sleep(1);
         printf ("\n sname is :%s", name);
         printf ("\n tot is : %d", t);
         found =1;
        }
       }
   }
 if(found==0)
  printf ("\n sorry Rec is not found");
  getch();
}
```

Scanned by CamScanner

Data Structures

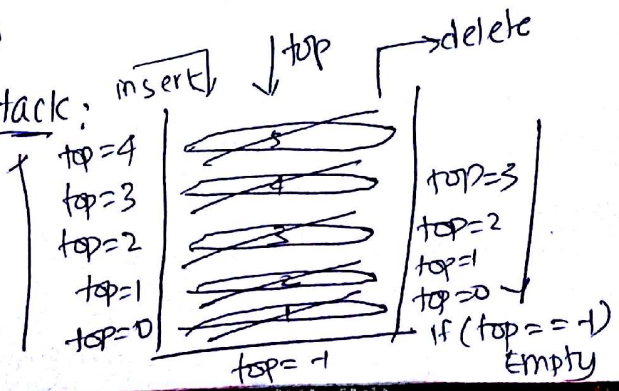
- It is a process of organising the data in a structured way of ~~pr~~representation (or) it is a logical ~~repr~~relationship between individual data elements
- The advantage of organising the data in a structured way of representation
  (1) Maniplating of the data becomes easier
  (2) Accessing the data becomes faster.
- Data structures are classified into two types
  (1) linear data structure
  (2) Non linear data structure.
- In linear data structure the data elements can be maintained and maniplated in a linear fashion.
  Ex: Array, stack, queue and linked list
- Non linear data structure the data elements can be maintained and maniplated in a non linear fashion
  Ex: Tree and graph

→ **Stack:**

- The stack is the collection of the organised elements in which new elements can be inserted and elements can be deleted at from only one end i.e, top end.
- The stack will follow the theory of LIFO accessing technique (last in first out)

→ **Graphical representation of stack:**

MAX = 5
If (top == MAX-1)
full



Scanned by CamScanner

→ stack related terminology:

— top:

Inserting the elements and deleting the elements will always from top end.

Most of the developers are used to take top value = -1 in order to know the status of the stack.

— Push:

Inserting an element into the stack is known as push. while inserting the elements into the stack the top value is to be raised by 1.

— Pop:

Deleting the value from element from stack is known as Pop. while deleting the va element from stack the top value is decreased by 1.

— stack overflow:

when there is no eteme space to pu push an element into the stack & when the top value == MAX-1 then the status of stack is overflow.

— stack underflow:

When there are no elements in the stack & when the top value == -1 the the status of stack is underflow

Scanned by CamScanner

→ **Program 215:**

Demo program for inserting the element into the stack by taking or Placing top-value = 0.

```
#define MAX 3
main()
{
 int s[MAX] ;i, top=0,
  clrscr();
  while(1)
   {
     printf("\n enter an element");
     scanf ("%d", &s[top]);
      top++;
       if (top > MAX-1)
        { printf ("\n stack is overflow");
           break; }
   }
  for(i=0;i<top;i++)
   {
    sleep(1);
    printf ("\n%d", s[i]);
    }
  getch();
}
```

Scanned by CamScanner

→ Program 21b:

Demo program for inserting the elements into the stack by placing top value = -1

```
#define MAX 3
main()
{
  int s[MAX], i, top=-1;
  clrscr();
  while(1)
   {
     top++;
      if (top>MAX-1)
        {
          printf("\n stack is OF");
           break;
          }
       else
         {
          printf("\n Enter an element");
          scanf("%d", &s[top]);
          }
     }
  for (i=0; i<top; i++)
   { sleep(1)
     printf("\n%d", s[i]);
   }
getch();
}
```

Scanned by CamScanner

→ **Program 21:**

Demo program for inserting and deleting an element from stack using functions.

```
#define MAX 3
int s[MAX], top= -1;

void insert (int x)
 {
   top++;
  s[top]=x;
  }
void delete ()
  {
    s[top]=NULL;
     top--;
    }
void list ()
  {
    for (i=0; i<=top; i++)
    { printf ("\n %d", s[i]); }
    }
void main()
 {  insert (10);
    insert (20);
    insert (30);
    delete();
    delete ();
    insert (40);
    list();
  }
```

Scanned by CamScanner

Operations on Stack

→ **Program 218:**

```
#include <stdio.h>
#include <conio.h>
#define MAX 5
int s[MAX], top=-1, i, ch, temp;
void push()
{ top++;
  if (top > MAX-1)
   {
     printf ("\n STACK IS OVERFLOW!");
   }
  else
   { printf ("\n enter an element ");
     scanf ("%d", &s[top]);
   }
}
void pop()
{ if (top<0)
    printf ("\n stack is UNDERFLOW!");
  else
   {
     temp= s[top];
     printf ("\n Deleted item is :%d", temp);
     s[top] = NULL;
     top--;
   }
}
```

Scanned by CamScanner

```
peek()
{
  if (top<0)
    printf("\n stack is UNDERFLOW");
  else
  {
    temp=s[top];
    printf("\n Element @top of stack :%d", temp);
  }
}
void list()
{
  if (top<0)
    printf("\n STACK IS UNDERFLOW");
  else
  {
    printf("\n Elements in stack");
    for (i=0; i<=top; i++)
    {
      sleep(1);
      printf("\n %d", s[i]);
    }
  }
}
void main()
{
  do
  {
    clrscr();
```

Scanned by CamScanner

```
printf("\n operations on stack");
printf("\n==========");
printf("\n 1. PUSH \n 2. POP");
printf("\n 3. PEEK \n 4. List");
printf("\n 5. EXIT");
printf("\n ==========");
printf("\n enter ur choice = ");
scanf("%d", &ch);
  switch(ch)
  {
    case 1:
      push();
      break;
    case 2:
      pop();
      break;
    case 3:
      peek();
      break;
    case 4;
      list();
      break;
    case 5:
      exit(1);
    default:
      print("\n Invalid choice");
  }
  getch();
} while(1)
```

Scanned by CamScanner

Queue

It is a non-primitive linear data structure in which new elements can be inserted from one end and the elements will be deleted at from another end. The queue will follow the theory of FIFO

FIFO (First In First Out)

Queue related technology:

1) rear: Inserting the elements into the queue will always be from backside (rear side).

While inserting the elements into the queue the rear value to be increased.

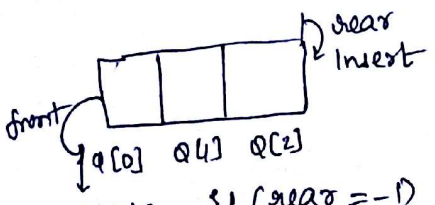
Steps for inserting the elements into the queue:

1) Increase the rear value by 1.
2) Ensure rear value is '0' or not.
3) If rear = 0 make front = 0. If not

ensure queue is overflow or not by using if (rear > max-1) display queue is overflow otherwise insert the element at Q[rear$].

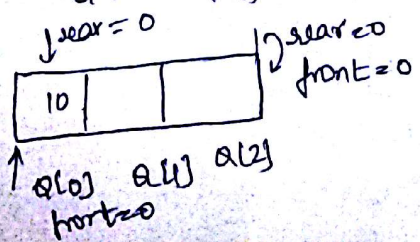
max = 3 rear = -1

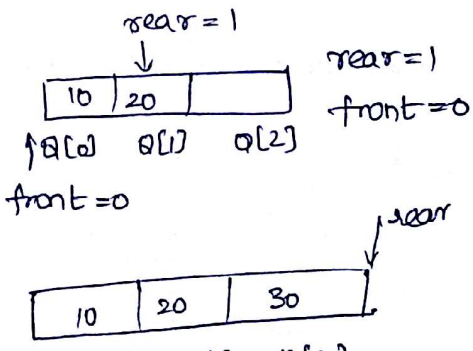
Q[max], front = -1

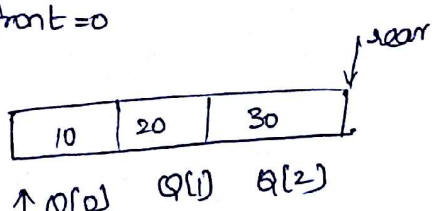


Delete if (rear = -1)

Q is underflow.







front = 0 if (rear = max-1)

Q is overflow.

```
list() {
    for(i = front; i<=rear; i++)
    printf("\n %d", Q[i]);
}
```

Scanned by CamScanner

Note: Elements are always presented between front and rear

Thus, loop through front to rear.

front: Deleting the element from the queue will always takes place from front position.

→ while deleting the element from the queue front value to be increased by 1

Steps for deletion:-

→ Ensure whether queue is empty or not by taking rear<0

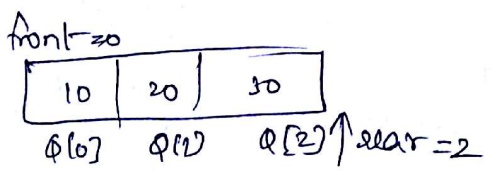
```
if (rear<0)
Pf("Q is Uf");
```

→ If Q is not underflow then ensure that the queue is having only element or not. if (rear==front)

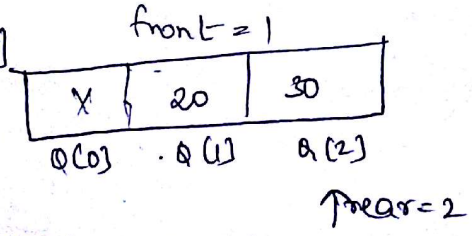
→ If both are same delete the element existed at Q[front] and make front=-1 and rear=1

→ If rare and front not equal then we have to delete the element existed at Q[front] and increase front value by 1.

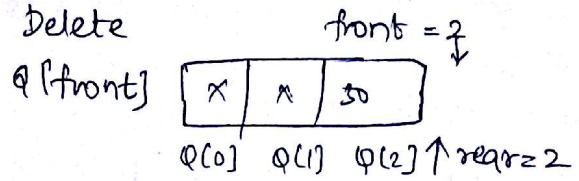
Q[MAX], rear=-1, front=-1

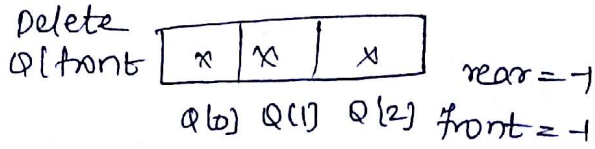


Delete Q[front]



Scanned by CamScanner





→ Queue overflow: when there is no space to push an element into queue or when rear value is greater than MAX-1 then status of Queue is overflow.

→ Queue underflow:- when there are no elements in the queue or when rear==-1 there status of Queue is underflow.

→ Program 2(a)

```
#include <stdio.h>
#include <conio.h>
#define MAX 5
int q[MAX], rear=-1, front=-1, temp, ch, i;
void insert( )
{ rear++;
    if (rear==0)
     front=0;
    if (rear>MAX-1)
      printf ("\n Queue is overflow");
    else
    { printf ("\n Enter an element");
      scanf ("%d ", &q[rear]);
    }
```

Scanned by CamScanner

```
void delete()
{ if (rear< 0)
    printf ("\n Queue is underflow");
    else if (rear== front)
      {
        temp = q [front];
        printf ("\n Deleted item is : %d", temp);
        q[front]= NULL;
        rear = -1;
        front = -1;
      }
    else
      {
        temp= q [front];
        printf ("\n Deleted item : %d", temp);
        q [front] = NULL;
        front ++;
      }
}
void list()
{
  printf("\n Element in QUEUE \n");
  for (i=front; i<=rear; i++)
    {
      sleep(1);
      printf("%d <- ", q[i]); }
}
```

Scanned by CamScanner

```
void peek()
 {
   if (rear<0)
   printf("\n Queue is underflow");
   else
     {
      temp= q[front];
      printf("\n front element of Queue : %d",temp);
     }
 }
void main()
 {
   do
     {
      clrscr();
      printf("\n OPERATIONS ON QUEUE");
      printf("\n==========");
      printf("\n 1.Insert \n 2.Delete");
      printf("\n 3. List  \n 4.Peek");
      printf("\n 5.Exit");
      printf("\n==========");
      printf ("\n Enter ur choice : ");
      scanf("%d ", &ch);
      switch(ch)
       {
        case 1:
          insert();
          break;
        case 2:
          delete();
          break;
```

Scanned by CamScanner

```
        case 3:
            list();
            break;
        case 4:
            peek();
            break;
        case 5:
            exit(1);
        default:
            printf("\n Invalid choice");
    }
getch();
}while(1);
}
```

Scanned by CamScanner

Searching Techniques

- The techniques which are used for searching the elements in the list.
- In programming we have two types of searching techniques
  1. linear searching (sequencial searching)
  2. binary searching

→ linear search:

- In the linear search th-each element in the list must be compared with the search number and the process need to be start from the beginning and continue the searching techniques till the desired element is found or till the end of list elements.

Program 220:

```
#include<stdio.h>
#include<conio.h>
#define MAX 5
void main()
{
  int a[MAX], sn, i, found=0;
  clrscr();

  for (i=0; i<MAX; i++)
   {
     Printf("\n Enter an element");
     scanf("%d", &a[i]);

   printf("\n Enter an element for search");
   scanf("%d", &sn);
   for (i=0; i<MAX; i++)
  {
    if (a[i]==sn)
```

Scanned by CamScanner

```
{
    found = 1;
    printf("\n Element is found");
    break;
  }
}

if (found == 0)
  printf("\n Sorry Element not found");

getch();

}
```

→ Binary search:

- In the binary search all the list elements must be in a sorting order.
- In the binary search enter list is divided into two partitions based on the approximate middest index value
- if (sn < a[mid]) then make the system to search towards down direction by setting high value = mid-1.
- if (sn > a[mid]) then make the system to search towards up direction by setting low value = mid+1
- if (sn = a[mid]) then that is the element required to find

→ Program 221:

```
#include <stdio.h>
#include <conio.h>
#define MAX 5
void main()
{ int a[MAX], sn, i, found=0, low, high, mid;
  clrscr();
```

Scanned by CamScanner

```
for (i=0; i<MAX; i++)
  {
    printf ("\n Enter an element");
    scanf ("%d", &a[i]);
  }
printf ("\n Enter an element for search");
scanf ("%d ", &sn);
low=0;
high=MAX-1;
while (low<= high)
{
   mid=(low+high)/2;
   if (sn<a[mid])
     high= mid-1;
   if( sn>a[mid])
     low= mid+1;
   if( sn==a[mid])
     { found=1;
       break;}
}
if (found==1)
  printf ("\n element is found");
else
  printf ("\n Element is not found"),
  getch();
}
```

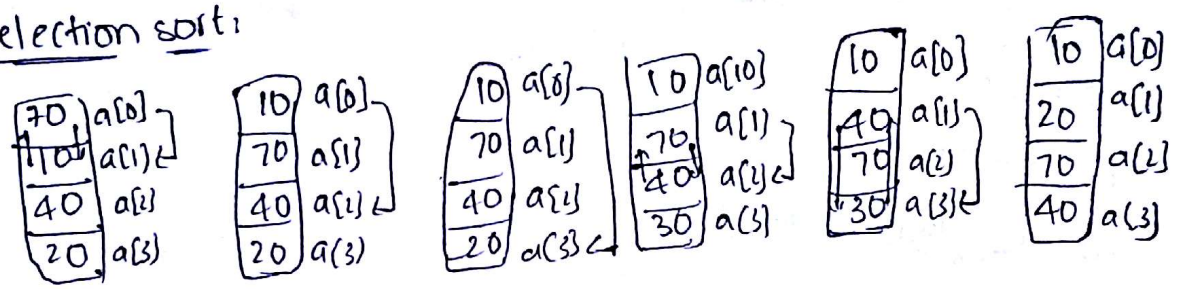
Scanned by CamScanner

→ Sorting Techniques :

The techniques which are used for making the list elements in a sorting order popularly we have 2 popular sorting techniques:

1) selection sort
2) Bubble sort

Selection sort:

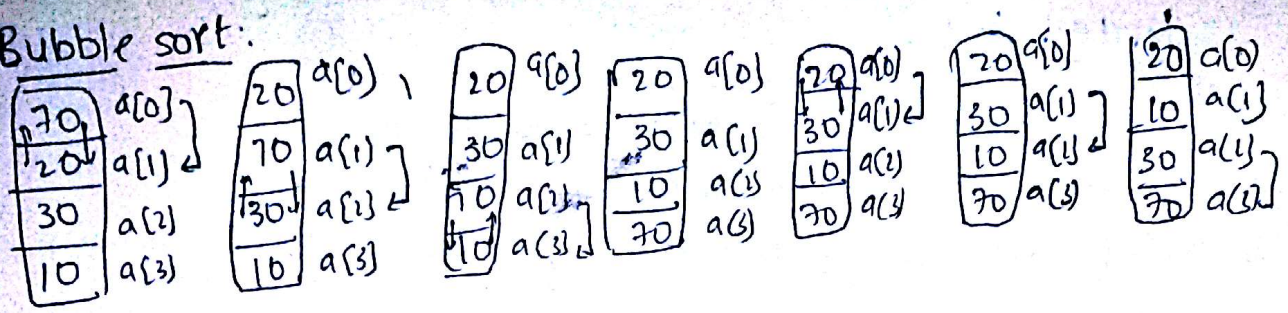


→ **Program 222:**

```
#define MAX 5
int a[MAX],i,j,temp;
void sort (int *x[])
{
  for (i=0; i<max; i++)
   { for (j= (i+1);j<max;j++)
     { if(x[i]>x[j])
       { temp=x[i];
         [i] = x[j];
         x[j]=temp;
       }
     }
   }
void main( )
{ clrscr();
  for (i=0; i<max; i++)
  { printf ("\n Enter an element",
    scanf("%d", &a[i]); }
  sort(a);
  pf("\n Elements after sorting");
  for (i=0) i<max; i++)
   { sleep();
     printf ("\n %d ", a[i]; }
  } getch();
```

Scanned by CamScanner

Bubble sort:



→ **Program 223:**

```
#include <stdio.h>
# include < conio.h>
# define MAX 5
void main ()
{ in a[MAX] , i , j , temp;
  clrscr();
  for (i=0; i<MAX; i++)
  { Printf ("\n Enter an element");
    scanf ("%d", &a[i]); }
  for (i=0, i<MAX; i++)
  { for (j=0, j<MAX; j++)
    { if (a[j]>a[j+1])
      { temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
      }
    }
    sleep(1);
  printf ("\n elements in sorting");
  for (i=0; i<max; i++)
  { sleep(1);
    printf("\n %d", a[i]); }
  getch();
}
```

→ **NOTE:** In order to make the list elements in descending order then we have to replace the following statements

In s.sort if (a[i]>a[j]) with if (a[i]<a[j])

B.sort if (a[j] > a[j+1]) with if (a[j]<a[j+1])

Scanned by CamScanner

linked list:

→ list is a collection of organised elements.

→ Array is the best example for the list

→ Arrays are having the following limitations

(i) We must specify the size of an array intially in the program before compilation.

(ii) It doesnot allow you to increase or reduce the size of an array during the program ~~the~~ execution

(iii) If we enter the data less than the specified capacity, possibility for wastage of ~~ex~~ memory

- Due to above inmilations arrays are not recommended to use of ~~so-pre~~ supplied to develop live applications.

- In order to eliminate the above limitations, we need to make use of linked list

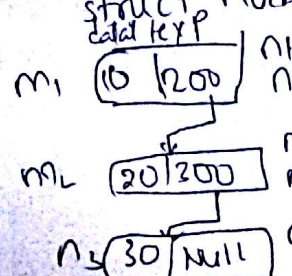
- In order to work with the linked lists we must required a spelficel self-refreshtional structure (a structure which is using it self with minimum of two number (or) imotta member

1) Data member 2) Pointer variable (* next)

- Data member to hold the blabble data. Next member to

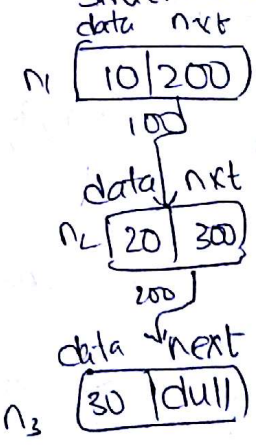
- Two hold the expl.ra to hold the adress of the next record

```
struct node{
    int data;
    struct node* next;
};
struct node n1,n2,n3;
```



n1: data=10
n2. tani
n1. data=20;
n2. next=&n3;
n1. data = 30;
n2. n3.next=NULL;

```
struct node *n1,*n2,*n3;
```



n1->data=10;
n1->next=&n2;
n1->data=20;
v2=>next=&m1
n3->data=30
n3->next=NULL;

Scanned by CamScanner

→ **mallac:**

It is a dynamic allocation function used to allocate the memory dynamically during the program execution.

```
int * x;
syn: <PV> = (type cast * )malloc(size);
        x = (int* )malloc(10);
        x = (int*) malloc (sizeof (malloc()*5));

struct node *new;
    new = (struct node* ) malloc(4); //good
    new = (struct node* )malloc (sizeof (struct node));
```

→ usually whatever the memory has been allocated for the variable will be get destroyed automatically soon after terminating the program. But in some situations, we need to deallocate the memory for the variables during the program execution, we need to make use of function

free()

Syn: free(<pv>);

Eg: free (new);

→ **Program 224:**

```
main()
{
 struct node
 { int data;
  struct node * next;
  };
  struct node *new, *frt=NULL,* t, *lst , clrscr();
  do
   {
     new = (struct node *) malloc (sizeof (struct node));
```

Scanned by CamScanner

```
new-> next = NUL;
printf ("\n Enter Data");
scanf ("%d", &new -> data);
if (fst==NULL)
{ fst= new;
  lst= new;
}
  else
   {
    lst->next=new;
    lst=new;
   }
  printf ("\n Do you want another rec y/n");
 } while (getch()!='n');
 for(t=fst; t!=NULL; t=t->next)
   {
    sleep(1);
    printf("\n %d", t->data);
   }
   getch();
}
```

NOTE:

The structure pointer always holding the adress of the structure and is pointing to initial member of the structure

Operations on linked list:

Program 225:

```
#include<stdio.h>
#include <conio.h>
struct node { int data;
              struct node *next;
            };
```

Scanned by CamScanner

```
struct node *new, *first = NULL, *last, *temp; *d;
char  cnfm;
int i, ch;
create()
{
  do{
    new= (struct node *) mallac (size of (struct node));
     new->next= NULL;
     printf("\n Enter Data",
     scanf ("%d", &new -> data);
      if(first ==NULL)
        {  first=new;
           last = new;
        }
      else
       { last->next=new;
         last=new;
      printf("\n  Do you want another rec Y/n:");
        fflush (stdin);
      scanf ("%c", & cnfm);
       }
     while (cnfm= ='y'//cnfm='Y');
    }
    void list()
    {
     temp = first;
     printf (temp ==NULL)
     printf(" \n List Is Empty ");
    else
    { for (; temp!=NULL; temp= temp-> next)
```

Scanned by CamScanner

```
    {  sleep(1);
       printf("\n %d", temp -> data);
    }
  }
}
void count()
{
  int cnt=0;
   temp= first;
    if (temp= = NULL)
    pf("\n list is empty");
    else
    {
      for (; temp != NULL; temp= temp -> next)
      {
        sleep(1);
        cnt++;
      }
      sleep(1);
     printf("\n No. of nodes are : %d", cnt);
    }
}
void find ()
{ int srno, found = 0;
   temp= first;
   if (temp= =NULL)
   printf("\n list is Empty");
   else
  {  printf("\n enter SRNO");
     scanf("%d", &srno);
```

Scanned by CamScanner

```
for (; temp!=NULL; temp=temp->next)
{
    if (temp->data==smno)
    {
        found=1;
        sleep(1);
        printf("\n Data is found");
        sleep(1);
        printf("\n Data is: %d", temp->data);
        break;
    }
}
if (found==0)
    printf("\n sorry Data not found");
}
void modify()
{   int found=0, mrno, ndata;
    temp= first;
    if (temp=NULL)
printf("\n list is empty");
else
{ printf("\n Enter MRNO");
  scanf("%d", &mrno);
  for (; temp!=NULL; temp=temp->next)
  {  if (temp->data==mrno)
     {  found=1;
        sleep(1);
        printf("\n Data is found");
        sleep(1);
        printf("\n Enter new data");
        scanf("%d", &ndata);
        temp->data = ndata;
        sleep(1);
        printf("\n Rec is updated...");
        breaks;
```

Scanned by CamScanner

```
    }
  if (found==0)
    printf("\n Rec is not found");
  }
void insert()
{ new=(struct node *)malloc(size of(struct node))
  new->next=NULL;
  printf("\n Enter Data");
  scanf("%d", &new->data);
 INSERT:
   clrscr();
  printf("\n Insert operations");
  printf("\n ----------");
  printf("\n % First \n 2. Last");
  printf("\n ------");
  printf("\n Enter ur choice:");
  scanf("%d", &ch);
  switch(ch)
  { case 1:
     new -> next= first;
      first=new;
      break;
     case 2:
    last -> next=new;
    last=new;
    break;
    default:
      printf("\n Invalid choice");
       sleep(1);
      goto INSERT;
  }
```

Scanned by CamScanner

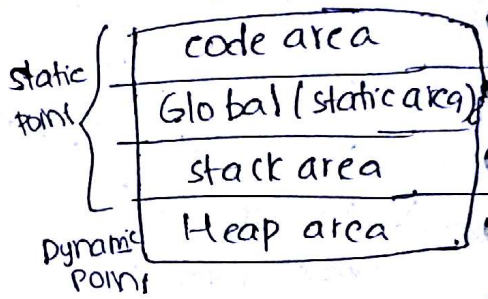
```
sleep(1);
printf("\n Rec is Inserted");
}

void delete()
{
 DELETE:
  clrscr();
  pf("\n DELETE operations");
  pf("\n ------");
  pf("\n 1. First \n 2. last");
  pf("\n ---");
  pf("\n Enter ur choice:");
  sf("%d", &ch);
  switch(ch)
  {
  case 1:
  d = first;
  first = first->next;
  free(d);
  break;
  case 2:
  for(temp=first; temp->next->next != NULL; temp=temp->next);
  d = temp->next;
  last=temp;
  last->next = NULL;
  free(d);
  break;
  default:
  pf("\n Invalid choice");
  goto DELETE;
}
```

```
sleep(1);
pf("\n Rec is deleted");
}
void main()
{
 do
  {
   clrscr()
  pf("\n operations on linked list");
  pf("\n ------");
  pf("\n 1. create \n 2. List \n 3. count");
  pf("\n 4. Find \n 5. Modify \n 6. Insert");
  pf("\n 7. Delete \n 8. Exit");
  pf("\n ------");
  pf("\n Enter ur choice ---:");
  sf("%d", &ch);
  switch(ch)
  {
   case 1:
    create();
    break;
   case 2: list(); break;
   case 3:
    count();
    break;
   case 4:
    find();
    break;
   case 5:
    modify();
    break;
   case 6:
    insert();
    break;
   case 7:
    delete();
    break;
   case 8:
    exit(1);
   default:
   printf("\n valid choice");
  }getch();
  }while(1);
}
```

Scanned by CamScanner

Storage Classes

- A variable is a space to store the data. In general, whenever you declare any variable. For every variable, the memory is allocated in the temporary memory area called RAM.
- In the RAM, we have 4 organisations



- Memory management is a process of allocating and diallocating the memory.
- Memory management is classified into two categories
  (1) Static memory management.
  (2) Dynamic memory management
- Static memory management is a process of allocating the memory for the variables of program by the time of compilation. In this case, the memory decisions will be done in the static point of RAM memory.
- Dynamic memory management is a process of allocating the memory for the variables by the time of execution. In this case, the memory decisions will be done in the heap organisation.
- Based on the programming requirements, we can select any of the organisation in the RAM memory for different purpose
- It is possible by using storage classes
- In C language we have 4 storage classes
  (1) automatic storage class
  (2) Static storage class

Scanned by CamScanner

3. External storage class
4. Register storage class.

→ NOTE :-
Every storage class is having its unique keyword, location, default value and lifetime.

SYNTAX: [storage class] <datatype> <name> [value]

1) Automatic storage class:

→ The keyword is auto
→ The location of these variables in the code area
→ Default value of these variables is garbage value.
→ Life time of these variables is within the scope.

→ Program 226:

```
main()
{
  auto int x;
  clrscr();
  printf("\n x=%d", x);
  getch();
}
```

output: Garbage value

→ Program 227:

```
void myfun(void);
main()
{
  clrscr();
  myfun();
  myfun();
  myfun();
  getch();
}
```

Scanned by CamScanner

```
void my fun()
{
  auto int x=10;
  printf("\n %d", x);
  x=x+10;
}
```

output: 10
10
10

→ Program228:

```
main()
{
  auto int x=10;
  {
    auto int x=20;
    {
      auto int x=30;
      printf("\n x = %d", x);
    }
    printf("\n x=%d", x);
  }
  printf("\n x=%d", x);
}
```

→ NOTE:-
By default, all the local variables are automatic storage class variables only
For the automatic storage class variables, memory will be re-intialised or re-allocated.

2) Static storage class:
- static in the key word.
- The location for these variables in the static area.
- Default value of these variables is '0'.
- Life time of these variables is persistent between function calls.

Scanned by CamScanner

→ Program 229:

```
main()
{
    static int x;
    clrscr();
    printf("\n x= %d", x);
    getch();
}
```

→ Program 230:

```
void my fun(void);
void main()
{
    clrscr();
    my fun();
    my fun();
    my fun();
    getch();
}
void my fun()
{
    static int x;
    printf("\n x= %d", x);
    x++;
}
```

NOTE:
For automatic storage class variable, everytime memory is re-initialised but static storage class variables, memory is allocated only once whenever that particular function is called and the state of the memory will be maintained till the function calls are completed

Scanned by CamScanner

External storage class:

→ The keyword for external storage class is "extern."

→ For these variables, memory is allocated in the global area.

→ Default value of this variable is zero.

→ Life of this variables is throughout the program

→ Program 231:

```
void my fun (void);
int x=222;
void main()
{
  extern int x;
    x=100;
    clrscr();
    printf("\n from main is x=%d", x);
    my fun();
    getch();
}
void my fun()
{
  sleep(1);
  printf("\n from my fun x=%d", x);
}
```

NOTE :-

If we declare both local and global variables, usually priority is given to local variables only

→ If we want to make the local variable as a global variable we need to make use of the key word

Scanned by CamScanner

→ one and only the variables which are not possible to initialise the value for the variable is extern

→ **Register storage class:**

→ In general, registers are the very small storage areas in the system. The register memories are recommended to use system programs.

→ If we store anything in the system registers then the performancy of these programs are faster---

Recursion function:

It is a process of calling a function to itself.

→ **Program 232:**

```
main()
{
 sleep(1);
 printf("\n Hello");
 printf("\n Good day");
 main()
}
```

output:

```
Hello
Good day
 :
 :
```

* **program 235:**

```
void my fun (int n)
 {
  sleep(1);
  if (n==0)
     return;
  else
  printf("\n %d", n);
  my fun(n-1);
 }
```

Scanned by CamScanner

```
Void main( )
{
   clrscr();
   my fun (18);
   getch()
}
```

10
9
8
7
6
5
4
3
2
1

Q) Define a function to find out factorial of n using recursion

Program 234:

Define a function of find out factorial of n using recursion

```
int n,f;
int fact (int x)
{
  if (x==0)
   return 1;
  else
   f= x* fact (x-1);
   return f;
}
void main()
{
  int r;
   clrscr();
   printf ("\n Enter a number");
   scan f(" %d", &n);
   r= fact (n);
   print f("\n fact is %d", r);
   getch();
}
```

Scanned by CamScanner