INDEX

NAME : T. Prashanth Reddy STD. : B.Tech SEC. : IT ROLL NO. : 12BIT0077

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		11599		
		CAT-II - Portion		
		1) Arrays		
		2) Functions		
		3) Algorithms		
		- Sine series		
		- Smallest divisor		
		- Square root		
		- Raising a no to large power		
		- Base conversion		
		QUIZ-II		
		- Arrays & functions		

ITE 101 (Problem solving using C):

Computer: commonly oriented machine particularly used for Trade and educational research

Von-Neumann Architecture of computer

machine language
- 0 - signals do not flow in the circuit (off state)
- 1 - signal flows in the circuit (ON state)

Hardware: physical parts of computer

Software: set of instructions

Firmware: combination of hardware and software (eg: mobiles, washing machines, Bluetooth devices)

Application software: software which we use and create

C → compiler

founder of C: Dennis Ritchie

Compiler is a system software which converts high level language to machine language

→ Interpreter is also a system software just like compiler

→ Interpreter compiles line by line but compiler compiles the whole program in one go.

: BASIC "(Beginers allpurpose symbolic code)
instruction
is an interpreter

Assembler: -

It is a system software which converts mnemonic codes to machine language. Assembly level language requires to remember a lot of instructions

INTRODUCTION TO 'C'

Features:

→ Robust language
→ Fast and efficient
→ structured language
→ Portable (capable of running in any system)
→ Extendability
→ middle level language (supports High level language & middle level language)

Structure of code:

→ Documentation section
→ Link section
→ Definition section
→ Global declaration section
→ main()
{
variable declaration part;
Executable part;

}

→ Sub programs

{
Variable declaration part;
executable part;
}

Documentation section:
non-executable part also called remarkable statement comments

// sample program (single line comment)

/* ---
--- → (multiple line comment)
--- */

Link section:

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

→ preprocessor (does some process before execution of program. It includes header file)

Some header files:

1) stdio.h → standard input output (supporting input functions of 'C')
   - scanf() → i/p f°n
   - printf() → o/p f°n

2) conio.h: console input output (defines o/i for monitor)
   - clrscr(); getch(); getche(); (key typed is reflected on the screen and then the screen vanishes)

return 0; (conio.h is not required)

→ 3) stdlib.h

exit()
malloc()
calloc()
free()
fflush()
atoi()

| system("Pause")
↳ it displays (press any key to continue)

4) math.h

5) string.h

strcpy() | strrev()
strcat | strcmp()
strlen()

→ Definition section:
used for #define. it is used to define a constant value

eg: #define Pi 3.14

→ Global declaration:
variables declared outside the main and other functions is called global variable

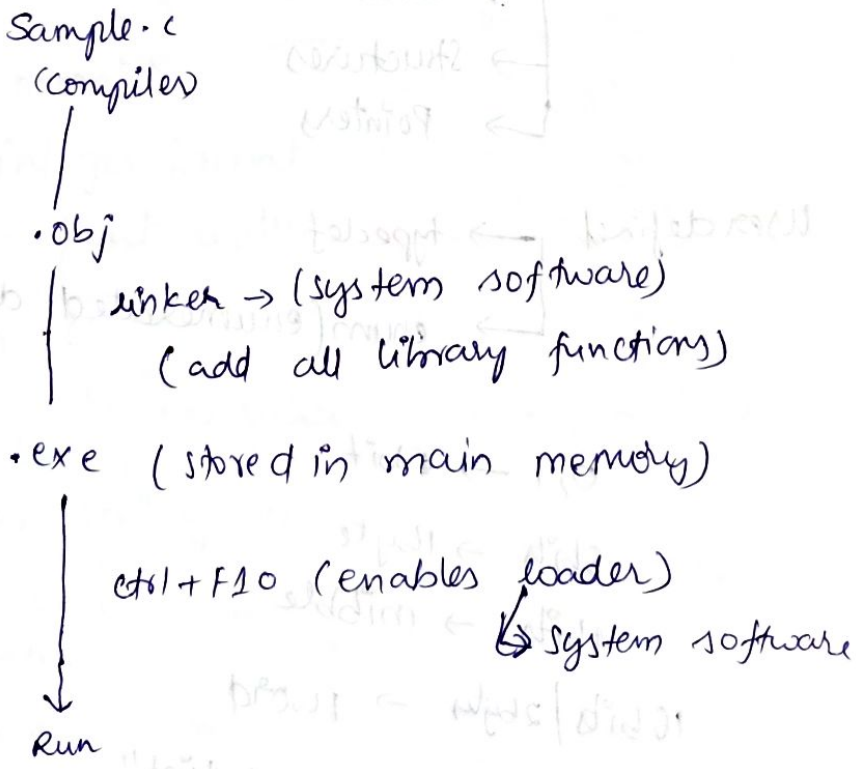
main() → is a function not a keyword

```
// sample program
#include <stdio.h>
#include <conio.h>
main()
{
printf("This is C programming");
getch();
}
```

Escaping sequences & applications:

\n

\t



Devc – 32 bit compiler

TurboC - 16 bit compiler

Data types:

- Primary / fundamental
- Secondary / Derived
- User defined

Primary data types
- integer (int) - 4bytes
- float (float) - 4bytes
- character (char) - 1byte
- double (double) - 8 byte
- void (void) - 0 byte

} for DevC++

Derived data Types
- Arrays
- functions
- structures
- Pointers

User defined
- typedef
- enum (enumerated data types)

0, 1 → 1 bit

8bits → 1byte

4bits → 1nibble

16bits / 2bytes → 1word

range of numbers that "int" can store $= -2^{n-1}$ to $2^{n-1}-1$

$= -2^{31}$ to $2^{31}-1$

To find minimum range and max range

int -MN;

→ when the program executes successfully then the value of 0 is returned when not, then (-1) is returned

→
```
// sum of two numbers
#include<stdio.h>
#include<conio.h>
main()                    //same as int main(void)
{
   int a=10, b=20, c;
   c=a+b;
   printf("The sum of two numbers %d+%d is %d",
                                   a,b,c);
   getch();
}
```

%d → address operator
%d = for integer format
%f = for float value
%c = for char value
%lf = for double value
%s = for string ..

→ sum of two numbers
```
#include<stdio.h>
#include<conio.h>
main()
{ printf int a,b,c;
  printf("Enter the value of a and b");
  scanf("%d%d", &a, &b);
```
→ locates the address of 'a' already allocated)
```
  c=a+b;
  printf("%d+%d=%d", a,b,c);
  getch();
}
```

Rules for declaring variables / identifiers:

→ → The first character of a variable should be a character or a underscore (_)

→ → Except the first character, the rest of the characters can be a digit or a letter or an underscore (_)

→ No keywords can be used as variables

→ No white spaces or blank spaces can be used

→ No other special characters are not allowed

→ The variables can be of any length but only first 31 letters are considered by the compiler.

NOTE:- constants
Keywords
variables
operators
special characters are called as 'C tokens'
(also called as "lexical elements")

→ WAP to exchange the values of two variables using a third variable and without using a third variable

a) using a third variable:

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a,b,c;
printf("Enter values of 'a' and 'b');
scanf("%d%d", &a, &b);
printf("The values of 'a' and 'b' before swaping %d and %d", a, b);
c = a;
a = b;
b = c;
printf("The values of 'a' and 'b' after swaping is %d and %d", a, b);
getch();
}
```

b) without using a third variable :-

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a,b;
printf("enter the values of a and b.");
scanf("%d %d",&a,&b);
printf("The values of 'a' and 'b' before swaping=");
Scanf
   a = a+b;
   b = a-b;
   a = a-b;
```

H.W

1. which function is used to update reminder value after dividing a float

A. ~~fmod~~. fmod(a,b) (supported by header file math.h)

2. which special symbol can be used in variable list

A. '_' (underscore)

3. By default compiler treat a real number

A. as double - to get more accuracy

4. What is the output of the following

```
#include<stdio.h>
int x=40;
int main(){
   int x=20;
   printf("%d",x);
}
```

A. Here x=40 is a global variable (since it is not in main function)
int x=20; is a local variable

→ local variable has higher priority than global variable

→ Hence output is 20.

5. A float is 4 byte wide and double is 8 byte wide

A. True

6. Size of datatype may vary from one platform to another

A. True (For Dev++ int = 4 byte for other —)

7. Global variables are variables available to all functions.

A. TRUE

8. Does there exist a mechanism by the way of which global variables is available to some function but not other functions

A. NO.

9. What format precifies are used to ~~print~~ scan the values of float and double

A. %f and _____

10. what is preprocessor directive

A #

11. A preprocessor is a msg from a programmer to the pre-processor

A TRUE

12. Every 'c' program will contain atleast one preprocessor directive.

A. FALSE

B. The preprocessor can trap simple errors like missing declaration, missing comments or miss match of braces

A FALSE

14.

A

15.

A.

16.

A.

17.

A

18

A

19.

A

20

A

14. A pre-processor directive is a msg from a compiler to linker

A. FALSE

15. One pre-processing is over and the program is sent for compilation the back slashes are removed from expanded source code

A. TRUE

16. which header files should be included for use of the functions like malloc(), calloc()

A. stdlib.h

17. what is the range of "character" data values

A. -128 to +127

18. Every executable program in 'C' must contain

A main function

19. A program block is enclosed with pair of

A. Curly braces

20. what is the output of

```
main()
{ printf("%d", out);
}
```

A Error

Operators in 'C'

1. Arithmetic Operator (+, -, *, /, %)
2. logical (&&, ||, !) True = 1 ; False = 0
3. Relational (>, <, <=, >=, ==, !=)
4. Conditional (? :) (Ternary operator)
5. Assignment (=)
6. Increment and decrement operators (++, --) (unary operator)
7. Bitwise operator (only for integer values)
   (&, !, ^) (<<, >>, ~)
8. special operator
   (, (comma), sizeof())

→ Sample program

```
#include<stdio.h>
main()
{  int a,b;
   scanf("%d %d",&a,&b);
   a>b? printf("A is greatest"): printf("B is greatest");
}
```

true — false

→ sample program

```
#include<stdio.h>
main()
{  int a,b,c;
   Scanf("%d %d %d",&a,&b,&c);
   a>b&&a>c? printf("A"): printf("B"): printf("C");
}
```

a>b ? a>c ?

Expressions:-

An expression is a collection of operators and operands

An expression can be prefix, infix (or) postfix. In prefix, the operator preceeds the operands. In infix, the operator is in between the operands and for postfix, the operator follows the operands

+ab → prefix

a+b → infix

ab+ → postfix

Sample program:-

```
#include<stdio.h>
main()
{
  int m=5, x, y;
  x = m++          // x = m (5), m = m+1 (6)
  print ("%d %d", x, m);
  y = ++m;         // m = m+1 (7), y = m (7)
  printf ("%d %d", y, m);
}
```

z = m++ + ++m   // m = m+1 ; z = m+m (16) ; m = m+1

r = ++x + y--;   // x = x+1 (6), r = x+y (6+7 = 13), y = y-1 (6)

Bitwise Op...

c = a&b

OR:

Bitwise operators:

and = &

or = |

negation = ~ → unary operator

left shift = <<

Right shift = >>

c = a & b

AND

a	b	o/p
0	0	0
1	0	0
0	1	0
1	1	1

a = 12 ; b = 6

c = 12 & 6

$(12)_{10} = (1100)_2$

$(6)_{10} = (110)$

$(12)_{10} \& (6)_{10}$ =

```
 1100
 0110
------
 0110
```

$\therefore c = 12 \& 6 = 4$

$(0110)_2 = (4)_{10}$

OR:

OR

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	1

c = 12 or 6 = 12

$(12)_{10} = (1100)_2$

$(6)_{10} = (110)_2$

$(12)_{10} | (6)_{10}$ =

```
 1100
 0110
------
 1110
```

$(1110)_2 = (14)_{10}$

c = 12 | 6 = 14

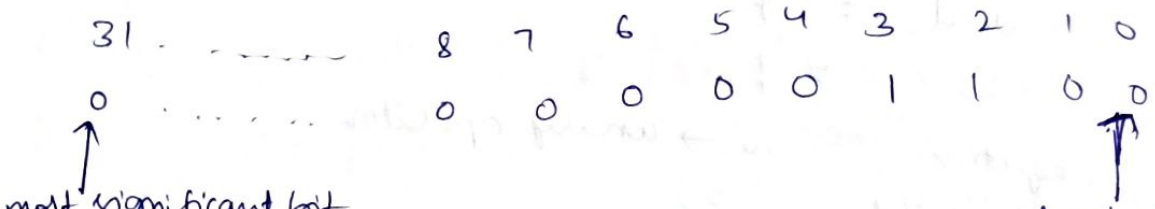
XOR

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	0

```
 1100
 0110
------
 1010
```

$(12)_{10} \wedge (6)_{10} = (10)_{10} = (1010)_2$

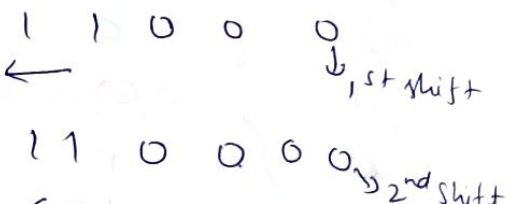
left shift:-



12 << 2

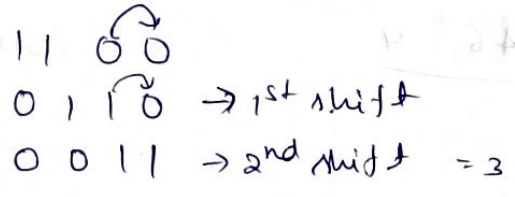
a << s → no. of shift bits

$\Rightarrow (12)_0 \rightarrow (1100)_2$



a << s output is $\boxed{a * 2^s}$

12 >> 2 = 3



a >> s output is $\boxed{\frac{a}{2^s}}$

only 31 shifts are possible

negation:-

c = ~a

↳ 2's complement of the output

i/p 12 o/p = -13

i/p 100 o/p = -101

Special Operators:-

"," and "sizeof()"

```
main()
{ printf ("%d %d %d %d %d %d", sizeof(int), sizeof(float),
          sizeof(char), sizeof(double), sizeof(void));
}
```

```
main()
{
    int x=10, y=20, z;    // seperator
    z = x, y, x+y;        //operator
    printf ("%d", z);
}
```

→ Comma has lowest precedence amongst all

Q. Write a program using conditional operators

1) To find a given number is odd or even

```
#include<stdio.h>
#include<conio.h>
main()
{
  int a, num;
printf ("Enter the number");
scanf ("%d", &num);
a = num%2;
(a==0) ? printf ("number is even") : printf ("number is odd");
getch();
}
```

2. To find the given year is leap year (or) not

```
#include<stdio.h>
#include<conio.h>
main()
{  int a, num;
   printf ("enter the number of days in the year");
```

```
scanf ("%d", &num);
a = num % 4;
(a == 2) ? printf ("year is not a leap year") : printf ("year
a leap year");
getch();
}
```

iii) To find a number is +ve (or) -ve

```
#include<stdio.h>
#include<conio.h>
main(){
  int num;
  printf("enter the number");
  scanf ("%d", &num);
  (num>0) ? printf ("number is positive) : printf ("number is
  negative);
  getch();
  }
```

iv) To find an input characters is vowel or consonant

Q) WAP to perform multiplication operation and division operation without using "*" and "/"

```
#include <stdio.h>
#include <conio.h>
main()
{ int num,a,b;
 printf("enter the number");
  scanf("%d",&num);
  printf("multiplying the number with 4\n");

  a=num<<2;
  printf("%d\n",a);
  printf("dividing the number with 4\n");

  b= num>>2;
  printf("%d\n",b);
  getch();
  }
```

Decision making and branching

- Conditional branching
  - if
  - if else
  - else if ladder
  - nested if
  - switch
- unconditional branching
  - goto

else if ladder:

Syntax:

```
if(condition)
{
  ---
}
else if (condition)
{
  ---
}
:
:
:
else
{
  ---
}
```

→ Sample Program:

```
main()
{
 char op;
  int a,b,c;
 printf("press +(or) -(or) *(or) /(or)%");
 scanf("%c %d %d",&op, &a, &b);
 if(op=='+'){
 {
    c=a+b;
   printf("sum is %d", c);
 }
  else if (op=='-')
 {
     c=a-b;
    printf("Difference is %d", c)
 }
 else if (op=='*')
 {     c=a*b;
  printf("product is %d", c);
}
else if (op=='/')
{
    c=a/b;
 printf("quotient is %d", c);
}else if (op=='%')
{  c=a%b;
 printf("remainder is %d", c);
}else
 printf("Entered wrong operator");
}
```

2.

Switch Case:

Syntax:

```
switch (variable or expression)
{
 case label1:
    ---
    ---
 case label2:
    ---
    ---
 case label3:
    .
    .
    .
 default:
    ---
    ---
}
```

→ Sample Program:

```
main()
{
  int op;
  int a,b,c;
  printf("Press 1. Add 2. Substract
    3. Multiplication 4. Division 5. Modulus \n");
scanf("%d%d%d", &op, &a, &b);
 switch(op)
 {
   case 1 : c=a+b;
        printf("sum is %d", c);
        break;
   case2 : c = a-b;
   printf("Difference is %d", c);
    break;
   case 3:   c = a*b;
   printf("Product is %d", c);
    break;
   case 4:   c=a/b;
    printf("Quotient is %d", c);
    break;
   case 5:
      c = a%b;
    printf("Remainder is %d", c);
    break
   default:
    printf("Entered wrong choice");
  }
}
```

Unconditional branching:

Syntax:

goto label;

```
main()
{
  int i=1;
loop: printf("%d\n", i);
      i=i+1;
      if(i<=10)
      goto loop;
}
```

```
loop:
==
==
goto loop;
```
} backward jump

```
goto loop;
loop
==
==
==
```
} forward jump

WAP to check whether the given no is divisible by '7' or not.

```
#include<stdio.h>
#include<conio.h>
main(){
int num, a, b;
printf("Enter the number");
scanf("%d", &num);
a = num%7;
if(a==0)
printf("number is divisible by 7"); else
printf("number is not divisible by 7"); getch(); }
```

2. WAP to check whether a candidate is eligible for voting or not
3. WAP to find the roots of an quad eqn
4. WAP using goto to find the sum of first 10 numbers

continuation See After 3 papers

ARRAYS

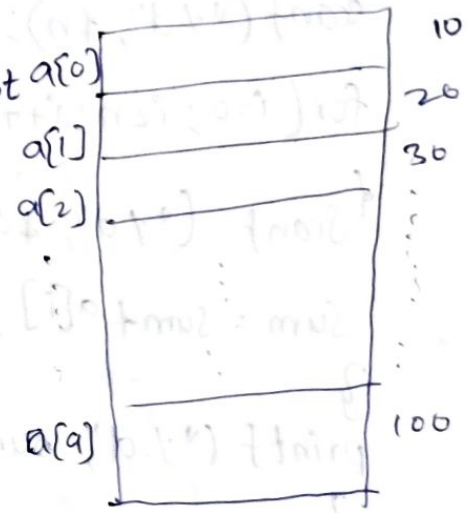
→ Arrays are ~~defined~~ derived data types which holds data items of same data type

Syntax:

datatype variable [size]

eg:

int a[10];
↳ index (or) subscript



→ Static Array:

→ Dynamic Array:

pointers

malloc() } <stdlib.h>
calloc() }

```
main()
{
  int i, a[100], n;
  scanf("%d", &n);
  for(i=0; i<n; i++)
   scanf("%d", &a[i]);
  for(i=0; i<n; i++)
   printf("%d", a[i]);
}
```

Q) Write a program to find sum of any array values

```
main()
{
int i, a[n], n, sum=0;
scanf("%d", &n);
for(i=0; i<n; i++)
{scanf("%d", &a[i]);
sum = sum+a[i];
}
printf("%d", sum);
}
```

Q) Scan for 'n' values and get an element 'x' and check whether the element is present in array or not. if present print 'element is present'. let the msg be printed only once.

```
main()
{
```

→ This is called "searching" & method is called "Linear (or) Sequential searching".

```
main()
{ int i, a[50], flag=0, x, n;
scanf("%d", &n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter the element to be searched");
scanf("%d", &x);
```

```
for(i=0; i<n ; i++)
{ if (a[i]==x)
 { flag =1;
  break;
 }
}
 if (flag==1)
 printf("element found");
 else
  printf("element not found");
 }
```

→

To find duplicates.

```
main()
{ int i, a[50], flag=0, x, n, c=0;
scanf("%d", &n);
[for(i=0; i<n; i++)
[scanf("%d", &a[i]);
[printf("Enter the element to be searched");
 scanf("%d", &x);
  for (i=0 ; i<n; i++)
{ if (a[i]==x)
 { flag=1;
   c++
  }
}
```

```
if (flag == 1)
printf ("Element found %d times", c);
else
printf ("element not found");
}
```

→ H.W

WAP to find the largest & smallest element in an array

SORTING:-

Arranging of data items either in ascending order (or) descending order

Bubble sort:

5
3
1
4
2

```
for(i=0; i<n-1; i++)        // i → no. of passes
{                           // j → no. of iterations within each pass
 for(j=0; j<n-i-1; j++)
 {
  if(a[j]>a[j+1])
  {
   temp = a[j];
   a[j] = a[j+1];
   a[j+1] = temp;
  }
 }
}
```

Bubble sort:

i=0

j=0	j=1	j=2	j=3	j=4
5	3	3	3	3
3)	5	1	1	1
1	1)	5)	4	4
4	4	4	5)	2
2	2	2	2	5

i=1

j=0	j=1	j=2	j=3
3	1	1	1
1)	3	3	3
4	4)	4)	2
2	2	2	4
5	5	5	5

i=2

j=0	j=1	j=2
1	1	1
3	3	2
2	2	3
4	4	4
5	5	5

i=3

j=0	j=1
1	1
2	2
3	3
4	4
5	5

4
5
2
1
3

Decision

while

for

do-while

while

syntax

initiali

while(

{

incr

FOR:

Syntax

For(ini

{

}

Decision making and looping:

while } entry controlled loops
for }

do-while } exit controlled loops

while

syntax

```
initialisation
while(condn)
{ ___
  ___
  increment }
```

sample Program:

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

FOR:

Syntax:

```
For(initialisation; condition; increment(or) decrement)
{ ___
  ___
}
```

sample Program:

```
main()
{
  int i, sum=0, n;
  scanf("%d", &n);
for (i=1; i<=n; i++)
{  sum = sum+i;
}
printf("%d", sum);
}
```

→ do-while

syntax:

```
initialisation;
do
{ ___
  increment;
} while (condition);
```

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

kbhit() → keyboard hit
↳ conio.h

Sentinel Controlled loops:

```
main()
{ int i=0;
  while(1);
  { printf("%d", i);
    i = i+1;
  }
}
```

```
main()
{ int i=0;
  while(!kbhit())
  { printf("%d", i);
    i = i+1;
  }
}
```

ALGORITHMS

Algorithm - design

Program - implementation

Algorithm: is a mathematical model of any solution

→ "Algorithm" is the logical representation of a program defined in definite steps.

Properties:

1. Simple
2. Concise and compact
3. Effective
4. Free of ambiguity

Algorithm for sum of two numbers:

→ Declare the variables a, b, c

→ Read the values of a and b

→ Compute c = a+b;

→ Display the value of c;

→ Algorithms are written using pseudo code) language

Pseudocode: It is a tool that allows adaptation to any high level language

→ PROCEDURE / PROGRAM / FUNCTION : SUM

```
BEGIN
VAR    A, B, C : INTEGER
READ   A, B
C := A + B
WRITE  C
END    SUM
```

Home - work :

1Q. WAP to find factorial of a given number

2Q. WAP to generate the fibbanocci sequence

0, 1, 1, 2, 3, 5, 8 - - -

3Q Sum of square of 'n' natural numbers

4Q Reversing the digit

5Q Sum of digit

6Q. No. of digits in a given no

7Q. To check the given no is palindrome or not

8Q. To check the given no is armstrong or not

9Q. To find the sum of +ve and negative no's until '0' is pressed. If '0' is pressed then next

10Q. To generate odd no. b/w 100 and 1000

11Q. To generate first 100 even no

12Q. To check the no. given is prime or not

130. To generate first 50 prime no's

Writing Algorithms for greatest.

```
PROGRAM   GREATEST
BEGIN
VAR   A,B:INTEGER
READ  A,B
IF    A.GTB THEN
WRITE "A IS GREATEST"
ELSE
WRITE "B IS GREATEST"
ENDIF
END   GREATEST
```

→ For finding summation (using while)

```
PROGRAM   SUMMATION
BEGIN
VAR   I,N, SUM: INTEGER
I:=1
SUM:=0
READ  N
WHILE  I.LEN DO
SUM:= SUM+I
I:I+1
END.WHILE
WRITE "THE SUM OF N NATURAL NOS IS", SUM
END   SUMMATION
```

using for:

```
PROGRAM SUMMATION
BEGIN
VAR
I:=1
SUM:=0
READ N
FOR I:=1 TO N INCR BY 1 DO
SUM:=SUM+I
END FOR
WRITE "THE SUM OF N NATURAL NO'S IS", SUM
END SUMMATION
```

using do ... while

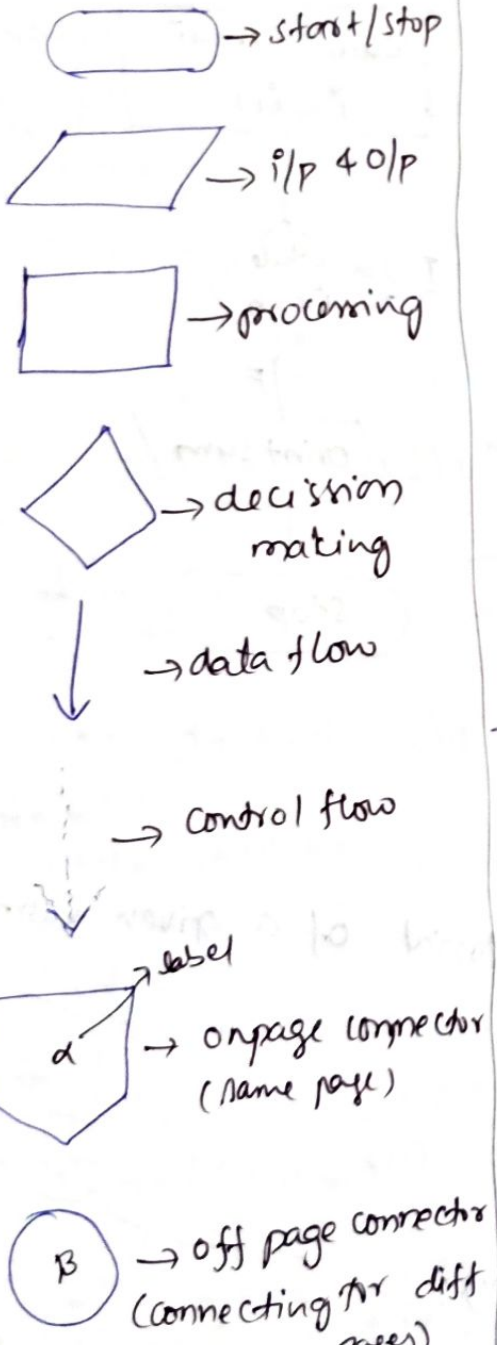
```
PROGRAM SUMMATION
BEGIN
VAR
I:=1
SUM:=0
READ N
REPEAT
    SUM:=SUM+I
    I:=I+1
UNTIL I.LE N
WRITE "THESUM OF N NATURAL NO'S IS", SUM
END SUMMATION
```

Executable pgm,

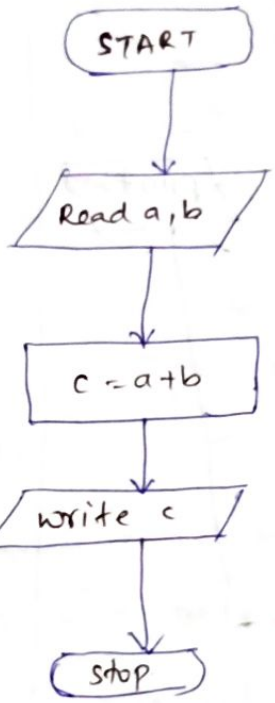
FLOW CHART

→ Pictorial representation of an algorithm is called as : flowchart

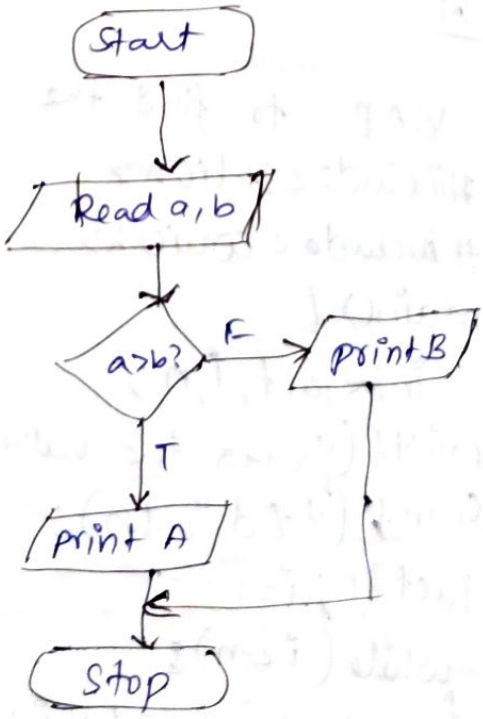
Symbols:



Flow chart for adding two numbers

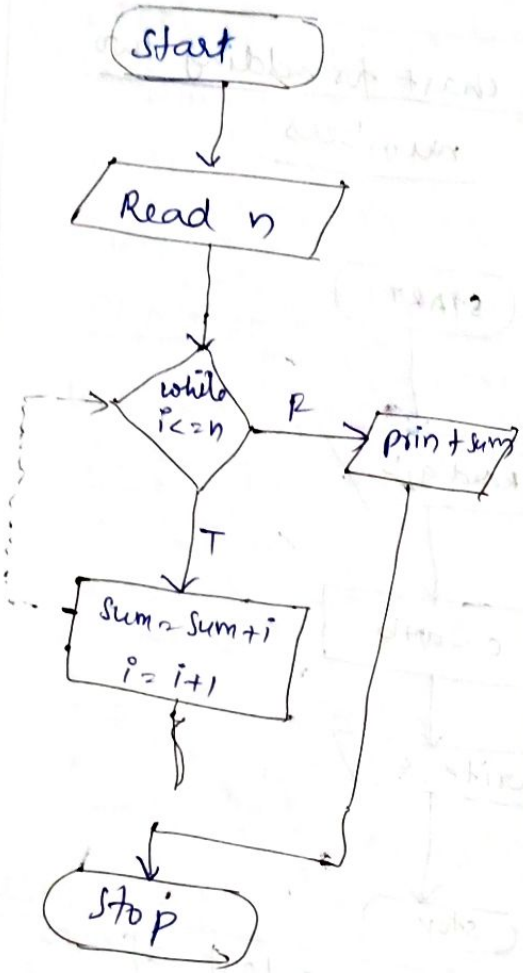


Flow testing greatest of two two numbers

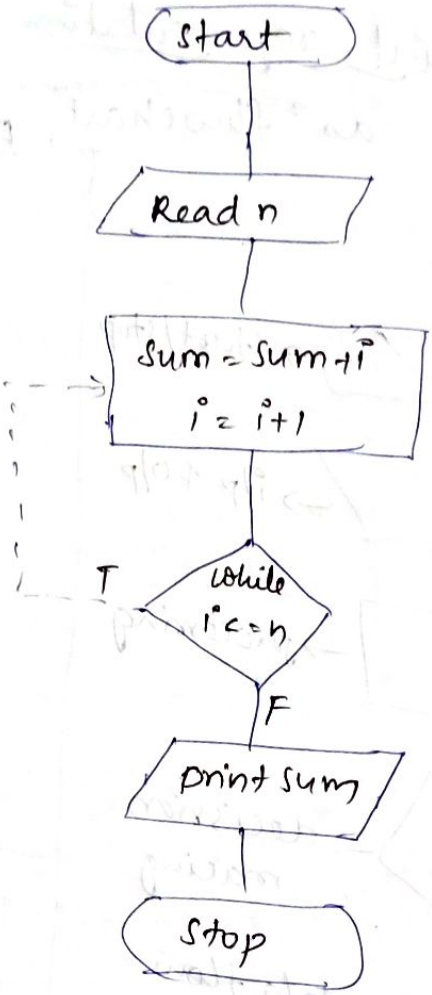


Looping flow chart

while



do while



H.W

1Q WAP to find the factorial of a given numbe

```
#include<stdio.h>
#include<conio.h>
main(){
  int fact, i, n;
printf("Enter the value of n");
scanf("%d", &n);
fact=1; i=1;
while(i<=n){
  fact = fact*i;
  i++;
}
getch(); }
```

29. WAP to generate the fibonacci sequence
0, 1, 1, 2, 3, 5, 8, 13, 21. . . .

```
#include<stdio.h>
#include<conio.h>
main()
{   int a=-1, b=1, c, i=0, n;
printf("enter the value of n");
scanf("%d", &n);
while(i<n)
{   c=a+b; a=b;
    b=c;
    i++;
    printf("%d\t", c);
}
getch();
}
```

30. WAP to find sum of squares of 'n' natural numbers

```
#include<stdio.h>
#include<conio.h>
main(){
    int sum=0, i=0, n;
printf("enter the value of n");
scanf("%d", &n);
while(i<=n)
{   sum = sum+i*i;
    i++;
}
printf("%d", sum);
getch();
}
```

4Q. REVERSING THE DIGIT

```
#include <stdio.h>
#include <conio.h>
main()
{
  int m, num, rev=0;
printf ("enter the number:");
Scanf ("%d", &num);
while(num!=0)
{
 m= num%10;
rev= rev*10+m;
num= num/10;
}
printf ("The reversed number is %d", rev);
getch();
}
```

5Q To find SUM of digits

```
#include <stdio.h>
#include <conio.h>
main(){
  int num, sum=0, m;
printf ("enter any number:");
Scanf ("%d", &num);
while (num!=0)
{  m = num%10;
  sum = sum+m;
  num = num/10;
}
printf ("sum of digits = %d", sum);
getch();
}
```

6. TO find no. of digits:

```
#include<stdio.h>
#include<conio.h>
main()
{
   int num, count=1, sum=0, m;
printf("Enter any number:");
scanf("%d", &num);
while(num!=0)
{ num = num/10;
  if(num!=0)
  count = count+1;
  else
  printf("no. of digits=%d\n", count);
  }
  getch();
  }
```

TO FIND GIVEN NO' IS A PALINDROME (OR) NOT

```
#include<stdio.h>
#include<conio.h>
main()
{
 int sum=0, m, num, temp;
printf("enter the number:");
scanf("%d", &num);
temp==num;
while(num!=0)
{
 m = num%10;
 num = num/10;
 sum = sum*10+m;
}
 if(temp==sum)
 printf("given number is a palindrome");
else
printf("given number is not a palindrome");
 getch();
}
```

Armstrong number:

Those numbers which sum of its digits to power of number of its digits is equal to that number are known as "armstrong numbers"

Ex: $153 = 1^3+5^3+3^3 = 153$

$1634 = 1^4+6^4+3^4+4^4 = 1634$

WAP to check whether given number is Armstrong number or not (for three digits)

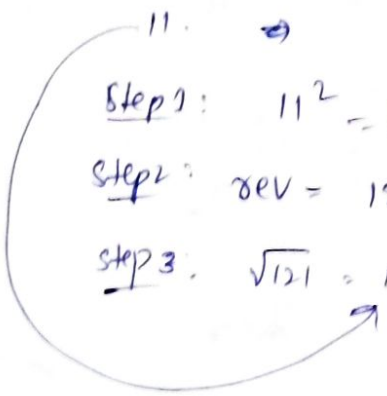
```
#include<stdio.h>
#include<conio.h>
main()
{
  int sum=0,m,num,temp;
  printf("enter the number:");
  scanf("%d",&num);
  temp=num;
  while(num!=0)
  {
   m=num%10;
   num=num/10;
   sum=sum+m*m*m;
  }
  if(temp==sum)
  printf("given number is Armstrong number");
 else
 printf("given number is not an Armstrong number");
 getch();
}
```

Strong numbers:

Sum of its digits factorial is equal to the number

$145 = 1! + 4! + 5!$

Adam's no:



10Q. To generate odd numbers b/w 100 and 1000

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,r,n;
    i=100; n=1000;
    while(i<=n)
    {
        r=i%2;
        if(r!=0)
        printf("%d\t",i);
        i++;
    }
    getch();
}
```

11Q. To generate first 100 even numbers

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,r,n;
    i=1; n=100;
    while(i<=n)
    {
      r=i%2;
      if(r==0)
      printf("%d\t",i);
      i++;
    }
    getch();
    }
```

To check whether the given year is leap year or not

Definition of leap year:-

Rule 1: A year is called leap year if it is divisible by 400

For Ex: 1600, 2000 etc leap year while 1500, 1700 are not leap year.

Rule 2:

If year is not divisible by 400 as well as 100 but it is divisible by 4 then that year are also leap year.

For Ex: 2004, 2008, 2012, 1012 etc are leap year

Algorithm for finding a leap year

IF YEAR MODULER 400 IS 0

THEN leap_year

ELSE IF year MODULER 100 IS 0

THEN not leap-year

ELSE IF year moduler 4 IS 0

THEN leap-year

ELSE

not - leap year

program using conditional operators:

```
#include<stdio.h>
#include <conio.h>
main()
{
   int year;
   printf("enter the year");
   scanf("%d", &year);
   (year%400==0)?printf("It is a leap year"):(year%100==0
?printf("It is not a leap year"):(year%4==0)?printf
("It is a leap year"):printf("It is not a leap year");

   getch();
}
```

To check whether the given number is prime number or not.

```
#include<stdio.h>
#include<conio.h>
main(){
    int num, i=2, count=0;
printf("enter the number");
scanf("%d", &num);
while(i<=num/2){
if(num%i==0)
count++;
i++;
}
if(count==0 && num!=1)
printf("number is prime");
else
printf("number is not a prime number");
getch();
}
```

To print prime numbers from 0 to 300.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
int i, j;
i=1;
while(i<300){
j=2;
while(j<sqrt(i)){
if(i%j==0)
break;
```

```
else {
    j++;
    continue;
  }
}
if( j> sqrt(i))
 printf ("%d\t", i);
i++;
}
getch();
}
```

Program to check the given number is strong number

without functions

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i, num, copy=0, r, f=1, sum=0;
    printf("enter the number");
    scanf("%d", &num);
    copy=num;
    while(num!=0)
    { int f=1;
     r=num%10;
     i=1;
    while(i<=r)
    { f=f*i;
     i++; }
     sum=sum+f;
     num=num/10;
    }
    if(copy==sum)
    printf("number is a strong number");
    else
    printf("number is not a strong number");
    getch();
    }
```

With functions

```
#include<stdio.h>
#include<conio.h>  --> int fact(int r);
main()
{
    int i, num, r, copy=0, f=1, sum=0;
    printf("enter the number");
    scanf("%d", &num);
```

```
copy = = num;
while (num != 0)
{ r = num % 10;
  sum = sum + fact(r);
 num = num/10;
}
if (copy = = sum)
printf ("number is a strong number");
else
printf ("number is not a strong number");
getch();
return 0;
}
int fact (int n)
{
 int i=1, f=1;
for (i=1; i<=n; i++)
{
f = f*i;
}
return f;
}
```

Find the max element in an array

```
main()
{
  int a[25], i, n, max;
  scanf ("%d", &n);
  for(i=0; i<n; i++)
  scanf ("%d", &a[i]);
  max = a[0];
  for (i=1; i<n; i++)
  { if (a[i] > max)
      max = a[i];
  }
  printf ("%d", max);
}
```

→ To get an array and printing it in the reverse order

Ex:

input
4
5
2
3
1

output
1
3
2
5
4

Logic

swaping

a[1] → a[n]

a[2] → a[n-2]

```
main()
{
  int a[5], i, n, c;
  scanf ("%d", &n);
  for
```

```
for (i=0; i<n/2; i++)
{
  c = a[i];
  a[i] = a[n-i-1];
  a[n-i-1] = c;
}
```

"divide and conquer" policy

Binary Search:-

→ binary search is efficient only if the input is in sorted order

key = 18

→ To find the mid value

0	1	2	3	4	5	6	7	8	9
11	12	13	14	15	16	17	18	19	20

```
low = 0;
high = n-1;
while (high >= low)
{
mid = (low + high)/2
if (a[mid] == key)
{ printf ("element found");
  break;
}
else if (key > a[mid])
low = mid + 1;
else
high = mid - 1;
}
```

mid = 0+9/2 = 4 (5) = 4

a[mid] = 15 ≠ key = 18

→ low = 4+1 = 5

mid = (5+9)/2 = 7 = key

→ H.W

1) Convert decimal number to binary

2) Removal of duplicates from ordered array and unordered array

Ordered array

1
1
2
2
3
4
5
5

unordered array

2
2
1
3
5
5
4
4

TWO DIMENSIONAL ARRAYS

Ex:- Matrix.

```
main()
{                    → rows   → columns
int a[10][10], n, i, j;
scanf("%d", &n);
for(i=0; i<n; i++)
for(j=0; j<n; j++)
scanf("%d", &a[i][j])
for(i=0; i<n; i++)
for(j=0; j<n; j++)
printf("%d", a[i][j]);
}
```

Program for the addition of two matrices

```
main()
{
  int a[10][10], b[10,10], i, j, n, c[10][10];
  Scanf("%d", &n);
  for(i=0; i<n; i++)
for (j=0 ; j<n; j++)
Scanf("%d", &a[i][j])
for (i=0; i<n; i++)
for (j=0 ; j<n; j++)
Scanf("%d", &b[i][j])
for(i=0 ; i<n; i++)
for(j=0 ; j<n; j++)
c[i][j] = a[i][j] + b[i][j];
for (i=0; i<n; i++)
for (j=0; j<n; j++)
printf ("%d", c[i][j]);
}
```

1Q) WAP To find the transformation of the matrix

2Q) WAP To print the diagonal elements of a matrix

3Q) WAP for the multiplication of matrix

```
main()
{
  int a[10][10], b[10][10], c[10][10], n, i, j, k;
  Scanf ("%d", &n);
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  Scanf ("%d", &a[i][j])
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  scanf ("%d", &b[i][j]);
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    c[i][j]=0;
  for (k=0; k<n; k++)
    c[i][j] = c[i][j] + a[i][k] * b[k][j];
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  printf ("%d", c[i][j]);
}
```

~~working of the loop~~ :-

Transpose of a matrix :-

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a[10][10], b[10][10], i, j, n;
    printf("enter the number of elements");
    scanf("%d", &n);
    printf("enter the elements of A matrix");
    for(i=0; i<=n; i++)
    {
     for (j=0; j<=n; j++)
    scanf("%d", &a[i][j]);
    printf("\n");
    }
    for (i=0; i<=n; i++)
    {
    for (j=0; j<=n; j++)
    b[i][j] = a[j][i];
    }
    printf("transpose of the matrix 'A' is b=\n");
    for(i=0; i<=n; i++)
    {
      for (j=0; j<=n; j++)
    printf("%d\t", b[i][j]);
    printf("\n");
    }
    getch();
}
```

c) WAP to find the sum of diagonal elements

a) WAP to fill upper right triangular matrix by '1' and lower right triangular matrix by '-1' and diagonals by '0' ⇒

$$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

b) WAP to ~~find~~ generate multiplication table of following format

	1	2	3	...	10
1	1	2	3	...	10
2	2	4	6	...	20
3	3	6	9	...	30
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.
10	10	20	30		100

WAP to convert decimal to binary

```
main()
{
    int n, a[100], i=0, j=0;
    printf("Enter a number");
    scanf("%d", &n);
    while(n!=0)
    {
        a[i] = n%2;
        n = n/2;
        i++;
    }
    for(j=i; j<=0; j--)
    printf("%d", a[j]);
    getch();
}
```

→ WAP for removal of duplicates from ordered array

```
main()
{ int i, j, n, a[100];
  printf("Enter the size of array");
  scanf("%d", &n);
  printf("enter values of array in sorted manner\n");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
  for(i=0, j=0; i<n-1; i++)
  {   if(a[i]!=a[i+1])
      {   j++;
          a[j] = a[i+1];
      }
  }
```

```
}
for (i=j+1 ; i<n ; i++)
 a[i]=0;
getch();
}
```

→ find the square root of a number.

```
int main(void)
{
  float g1, g2, m;
  printf("enter a number:");
  scanf("%f", &m);
  g2 = m/2; // initial guess all square roots from
                1-9 are close to their half value
  do
  {
    g1=g2; // g1 is previous guess
    g2 = (g1 + m/g1)/2; // g2 is current guess m/g1
                          is complementary
    printf("g2 = %f\n", g2);
  }
  while (fabs(g1-g2) > 0.0001);
  printf("Square root is = %f", g2);
  system(pause);
```

→ Find the smallest divisor:

```
int main(void)
{
    int n, d=3, s, sdivisor;
    scanf("%d", &n);
    if (n%2==0)
    sdivisor = 2;
    else
    {
        s = sqrt(n);
    while ((n%d!=0) && (d<s))
    d = d+2;
    if (n%d==0)
    sdivisor = d;
    else
    sdivisor = 1;
    }
    printf("smallest divisor is %d", sdivisor);
    system("pause");
}
```

→ nth power of number:-

```
int main(void)
{
    int n, x, p=1;
    printf("enter the values of x and n");
    scanf("%d %d", &x, &n);
    while (n>0)
```

```
{ if (n%2 == 1)
    p = p * x
  n = n/2;
  x = x * x;
}
printf ("%d", p);
system ("pause");
}
```

Assignment :-

1. WAP to Compute $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$ n terms

2. WAP to print the numbers that do not appear in the fibonacci. The no. of such terms to be printed should be given by the user

3. WAP to print the second largest no. in the given array

4. WAP to find the intersection of two array

5. WAP to remove duplicates from an ordered array

Conversion from decimal to binary number

```
#include <stdio.h>
#include <conio.h>
main()
{
  long int dnumber, quotient, n;
  int bnumber[50], i=1, j;
  printf("enter the decimal number");
  scanf("%d", &dnumber);
  quotient = dnumber;
  while (quotient != 0)
  {
  bnumber[i] = quotient %2;
  quotient = quotient /2;
  i++;
  }
  printf("binary number = ");
  for (j=i-1 ; j>0 ; j--)
  printf("%d", bnumber[j]);
  getch();
  }
```

Binary to decimal conversion

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
  long int decimalnumber=0;
  int i=1, j, binarynumber[50], n, k;
  printf("enter the number of elements");
  scanf("%d", &n);
```

```
printf("enter the binary number");
for(i=0; i<=n; i++)
scanf("%d", &binarynumber[i]);
for(j=i-1; k=0; j>=0; j--, k++)
decimalnumber = binarynumber[j] * pow(2,k)+decimalnumber;
printf("decimal number equivalent of the given
        binary number is");
printf("%d", decimal number);
getch();
}
```

1) To Compute the series $1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\cdots$ n terms

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i, j, k, x, fact, prod, n;
    float sum=0, term=0;
    printf("enter the value of x");
    scanf("%d", &x);
    printf("enter the value upto which series to be calculated");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    { fact=1;
      prod=1;
      for(j=1; j<=i; j++)
      {
        fact = fact*j;
        prod = prod * x;
      }
      term = (float) prod / (float) fact;
```

```
    sum = sum + term;
}
printf("the sum of the series is %.f", sum);
getch();
}
```

Functions are subprograms

- built in (or) library functions
  - ex: <math.h>, <ctype.h>, <stdlib.h>, <string.h>
- user defined
  - ex: main()

```
int sum (int, int);        → function prototype (or) function declaration
main()
{ int a, b, c;
  scanf ("%d %d", &a, &b);
  c = sum (a, b);          → function call
  printf ("%d", c);
  getch();
}

int sum (int x, int y)     → function heading (no semicolon)
{
  int z;
  z = x + y;               → function body
  return z;
}                          → function definition
```

(formal arguments)

(actual arguments)

→ If simple return is used it acts as closed curly braces

→ Stack:
A register in the ALU unit (Arithmatic logical) for function calls

→ The arguements which are passed by "function call" to the "function" are called "call by value"

→ In this only the copy of the values are passed any changes in the values in function that do not reflect in

the actual values of the variables

→ Call by Reference (or) address:

→ possible with "pointers"

* In this address (memory cell) is passed to the function. Any changes will be reflected in the original value

Categories of functions

1) No argument · no return
2) with argument no return
3) no argument with return
4) with argument with return

2) with argument no return:

```
void sum (int, int);
main()
{ int a,b,
  scanf ("%d %d", &a, &b);
  sum (a,b); → function call
  getch();
}
```

```
void sum(int x, int y)
{ int z;
  z = x+y;
  printf("%d", z);
}
```

3) No argument with return:

```
int sum(void);
main()
{ int c;
  c = sum(x, y);
  printf("%d", c)
  getch();
}
int sum(void)
{ int x, y, z;
  scanf("%d %d", &x, &y);
  z = x+y;
  return z;
  printf("%d", x+y);
}
```

M.W

1. Write a 'c' program to sort the given characters

Example of passing one-D array into functions

size

square bracket is must to indicate that we are passing array

```
int sum ( int [], int);
main()
{
   int a[50], i, n, val;
   scanf ("%d", &n);
   for (i=0; i<n; i++)
   scanf ("%d", &a[i]);
  val = sum (a,n);
  printf ("%d", val);
  }

int sum ( int a[], int n)
{
   int  i, tot = 0;
   for (i=0; i<n; i++)
   tot = tot + a[i];
   return tot;
   }
```

1st is optional & remaining should be given size

```
void addmatrix( int [][50], int [][50], int);

main()
{
    int a[50][50], b[50][50], i, j, n;
    scanf("%d", &n);
    for(i=0; i<n; i++)
      for(j=0; j<n; j++)
      scanf("%d", &a[i][j]);
    for(i=0; i<n; i++)
    for(j=0; j<n; j++)
    scanf("%d", &b[i][j]);
    addmatrix(a, b, n);
}
void addmatrix(int a[][50], int b[][50], int n)
{
    int k, l;
    for(k=0; k<n; k++)
    for(l=0; l<n; l++)
    printf("%d", a[k][l] + b[k][l])
}
```

a[0][0] + b[0][0]

Recursion

1. a function calling onto itself
2. Uses a data structure called stack
3. a conditional statement is required to e. from recursive function
4. solves complex problems easily by using "divide and conquer" strategy

Eg:

```
main()
{
  printf ("Recursion");
  if ( !kbhit())
   main();
  }
```

Finding factorial using recursion

```
int fact(int);
main()
{
  int val, n;
  scanf ("%d", &n);
  val = fact(n);
  printf ("%d", val);
  }
```

```
int fact (int m)
{ int f;
  if (m==1)
  return 1;
  else
  f = m * fact (m-1);
  return f;
}
```

printf("%d", m);

printf("%d", m);
printf("%d", f);

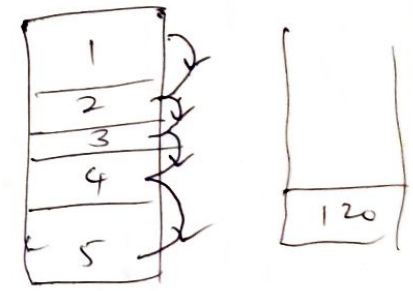
* a single return can return only a single value
* for returning multiple values we have to use pointers

any exp(or) equn gets converted into postfix expression and it is executed from right to left

linear datastructure

STACK - property of stack - last in first out

Ex:- wearing bangles



Q Home work

1. Using functions WAP to compute factorial of 'n'.
2. to find largest element in an array
3. To display diagonal elements
4. To perform transpose of a matrix
5. To print fibonacci sequence upto 'n' terms using recursion

Program to print the diagonal elements

```
#include<stdio.h>
#include<conio.h>
main()
{
  int a[10][10], i, j, n;
  printf("enter the number of elements");
  scanf("%d", &n);
  printf("enter the elements of A matrix");
  for(i=0; i<=n; i++)
  {
   for(j=0; j<=n; j++)
   scanf("%d", &a[i][j]);
   printf("\n");
  }
  for(i=0; i<=n; i++)
  {
   for(j=0; j<=n; j++)
   {
    if(i==j)
    printf("%d\t", a[i][j]);
   }
  }
   getch();
  }
```

STRINGS

```
main()
{
char name[100];
scanf("%s", name);
printf("%s", name);
}
```

→ no "&"

this will not read the string with space

To read string with space

<stdio.h> { gets();
puts();
functions are used

→ scanf("%[^\n]", name);
↳ not

except whatever given in the brackets all the things are accepted

→ scanf("%[abc]", name)
→ a, b, c are not accepted

→ scanf("%[~]");

STRING HANDLING FUNCTIONS

<string.h>

1) strcat - string concatenation = combining two strings
2) strcpy - string copy
3) strcmp - string compare
4) strlen - string length
5) strrev - string reverse
6) strchr - To search the values in a string
7) strstr - It will return the pointer value

1) strcat(str1, str2)

str1 = VIT ; str2 = university

str1 = VIT university

2) String Copy:

strcpy(str1, str2)

Syntax:
strcpy(destination_string, source_string, number of characters)

str1 = VIT ; str2 = university

str1 = university

3) String compare:

str1 = ABCD

str2 = abcd

* If the two strings given are equal then it will return '0'.

* It substracts the ASCII values of two strings

4) String length:

strlen(str1):

String length returns no. of characters in the string

strlen(VIT) : 3

5) String reverse:

strrev(str)

6) strchr

strchr(str1, I);

To search the values in a string

7) strstr

strstr(str1, str2);

⇒ it will return the pointer value

Q WAP to copy the string without using string handling function.

```
main()
{
char a[50], b[50];
int i=0;
gets(a);
while(a[i])
for
{
b[i]=a[i];
i++;
}
// b[i]='\0';
puts(b);
}
```

Q) WAP to read a string with spaces and print it without spaces

a) WAP to concatenate two strings

a) WAP to compare two strings

POINTERS

* derived data type
* a pointer provides away of accessing a variable without refering to the variable directly
* the mechanism used for this is the address of the variable
* Thus, a pointer variable is a variable that holds the memory address of another variable
* This variables are called as pointers because by storing an address they point to a particular value in a memory

Advantages:-

* makes efficient programs
* It is a powerful programming construct in 'C'

Disadvantages:

* difficult to understand
* errors produced by this is difficult to debug

Uses:

* used to call by address
* facilitating the changes made to a variable in the call function to become permanently available in the function from where it is called
* used to return more than one value from a function indirectly
* used to pass arrays and strings more conviniently from one function to another
* it communicates information about memory using malloc() function which return the location of free memory
* compiles coding faster and creates efficient code than any other derived data types such as arrays
* can be used to create complex data structures such as 'linked lists' and 'binary trees'

points to the value at address

indirection operator (or) dereferencing operator

```
main()
{
int a=5, *ptr;
ptr = &a;
printf ("%d", a);  → 5
printf ("%d", *ptr); → 5
printf ("%p", &a); → address of a
printf ("%p", ptr); → address of a
printf ("%p", &ptr); → address of ptr
}
```

* integer pointer should point to int address
* float pointer to float address

* The size of a pointer is always 4 bytes (depending on 0.5)

* memory address will be in "hexadecimal"

format

"%p", "%x" → are used to print the address of the variable

```
main()
{
int a=5, *ptr;
ptr = &a;
printf ("%d", a);  → 5
a=10;
printf ("%d", a);  → 10
```

```
*ptr = 15;
printf("%d", a);        → 15
}
```

```
main()
{
int a, b, *pa = &a, *pb = &b;
scanf("%d %d", pa, pb);
printf("sum = %d", *pa + *pb);
}
```

*(++p) — increment the memory address

++(*p) — increments the value stored in memory address

H.W

① WAP using pointers to exchange the value of two variables using & without using two variables

② WAP using pointer to compute factorial of n.

③ Differences b/w Arrays & Pointers

Swaping using functions:

```
void swap(int *, int *);
main()
{
    int x, y;
    scanf ("%d %d", &x, &y);
    printf ("values before swaping x=%d & y=%d", x, y);
    swap (&x, &y);
    printf ("\n values after swaping x=%d & y=%d",
                                        x, y);
    system ("pause");
}
void swap (int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Returning multiple values using pointers.

```
float compute(float, float *);
int main()
{
  float r, area, circum;
  printf("enter the radius in");
  scanf("%f", &r);
  area = compute(r, &circum);
  printf("Area = %f in", area);
  printf("circumference = %f in", circum);
  system("pause");
  }
  float compute(float r, float *p)
  {
   float a;
   a = 3.14 * r * r;
   *p = 2.0 * 3.14 * r
   return a;
   }.
```

Returning Pointers address

return data type

```
int *pointmax(int *, int *);
int main()
{
    int a, b, *p;
```

```
printf ("enter the values of a &b\n");
scanf ("%d %d", &a, &b);
p= pointmax (&a, &b);
printf ("The largest value is %d", *p);
system ("pause");
}
int * pointmax (int *x, int *y)
{
 if (*x > *y)
 return x;
 else
 return y;
 }.
```

---

* Array is a pointer constant

scaling

base adress + i * size of the data type

```
int main()
{  int a[5], i;
   printf ("enter array A\n");
   for (i=0; i<5; i++)
   scanf ("%d", &a[i]);
    for (i=0; i<5; i++)
    printf ("%d\t", *(a+i)); //a[i]
    system ("pause");
    }
```

when array is used the array name(a) holds the base address, the compiler adds the base address to subscript ix size of data type to get the offset (balance) address. This process is called 'scaling'. Hence, the subscript notation is converted to a pointer notation. use of pointer notation *(a+i) works fastly reducing the conversion time

26/10/2012

USER'S DEFINED DATA TYPES

STRUCTURES AND UNIONS

fflush → file flush

Keyword - struct

```
struct tag_name
{
    member1 ;
    member2 ;
    .
    .
};(list of variables separated by comma);
```

or

```
struct tag_name ( , , );
```

without tag-name :- Variable structure

including " " : Tagged structure

Type definition structure

↳ typedef

↳ can be given an alternative name (alias) for existing data type.

Access operators: Connects variables & tag name

- Dot operator
- →

* The extra space (memory) added by compiler (padding) is "Slack byte" in order to round the memory to multiples of "4"

diff b/w "UNION" & "STRUCTURES"

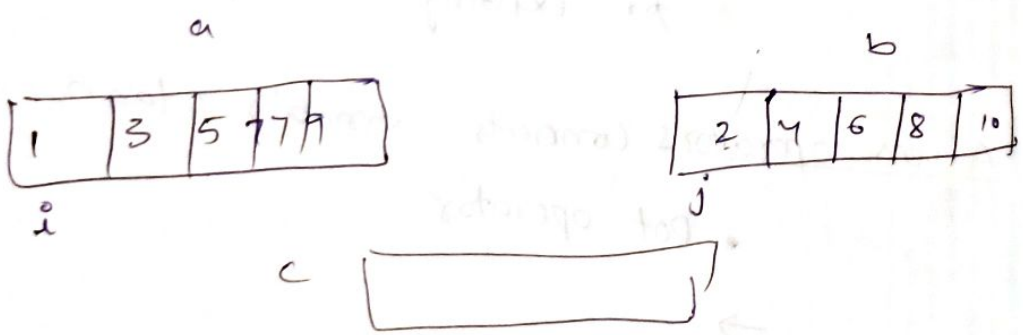
* Key word "UNION"
* difference in memory allocation
* only one member can be accessed at a time in union. (It is the choice of union) ↓ drawback
* In union whichever the size is most it is the size of array

Home Work:-

1Q. WAP to sort a list of ten names in alphabetical order

2Q. WAP to merge two arrays in sorted order so that if an integer is in both arrays and gets added to final array only once

3Q.



3Q. WAP to convert decimal to hexadecimal

4Q. WAP using user define function ~~to~~ to check whether the given no is strong or not

5Q. for hospital keeps, a list of blood donars in which each record has the following fields name, age, address., blood type.

6Q. WAP to create an array of structures to store all these details and print the list of all blood donars whose age is below 35. and blood type is ot.

FILES

read mode - To retrieve data

write mode -

append mode - to update file

FILE → <stdio.h>

→ inbuilt data structure in 'C'

→ To access file we should use pointer

Syntax: FILE * f1;

f1 = fopen("input", "w");
↑ mode
↳ opening a file

while(c = getchar() != EOF)
↳ end of file
(until we press ctrl+Z)

Comment for writing a character into a file = putc(c, f1);

fclose(f1); → close the file input

reading characters: putc, getc } file operators

getw, putw } reading integers

1. Difference between

   a. structures and Union

   b. structures and Arrays

2. Write a 'C' program to create records of a book store with following fields

Book name	Author	Edition	Book price
(string)	(string)	(integer)	(float)

3. Create a file structure called TEXT. Read the contents of text file character by character & the character read is a vowel write it to file called vowel otherwise, write it to a file called consonant.

---

structure

C++

→ SIMULA 67

→ SMALL TALK

→ Bjarne stroustrup – developed C++
– 1979

→ To solver larger problems

→ 'C' with classes

→ 'C++' superset of 'C'

OO approach:

Real world entity
↓ mapping
object
↓
data
functions
Interface implement

Features of OOP:

a) encapsulation
b) polymorphism
c) Inheritance

a) Encapsulation :-

program :

#include<iostream.h> → i/o stream of bytes

ipdev ← cin
opdevice cout
cerr
clog

scanf(...) → functions
printf(...)

cin >> → extraction → extracts from i/o device
operator

cout << → insertion " "

objects

stdin
stdout → c
stderr linked (used during program execution)

int	2
char	1
float	4
double	8
void	0

① #include<iostream.h>

void main()

{

cin >>

int a, b, c;

cout << "enter a";

cin >> a;

cout << "enter b";

cin >> b;

c = a + b;

cout << "sum of" << a << "and" << b

: In" << "is" << c; }

1) WAP to display the series 1 2 2 3 3 3

2) WAP to display the sum of the series $1+2+3+\ldots+n$

3) finding greatest of 3 numbers

1)
```
#include <iostream.h>
void main()
{
   int n, i;
   cin>> n;
   for(i=0; i<=n; i++)
   cout<< i;
   }
```

2)
```
#include <iostream.h>
main()
{
   int n, sum=0, i;
   cin>> n;
   for(i=0; i<=n; i++)
   sum = i+sum;
   cout<< sum;
   }
```

```
#include <iostream.h>
void main()
{
  for (int i=1 ;i<=4; i++)
  {
    for (int j=1 ;j<=i ;j++)
    {
      cout << "*" << j ;
    } // j
      cout << "\n";
    } // i
} // main.
```