S. No.	Date	Title	Page No.	Teacher's Sign. / Remarks

Date 28/09/21
Page 02

HISTORY OF C

* First language :- ALGOL :- Algorithmic language
  - mother of all modern programming language
  - 1960 international group.
* Second language :- BCPL :- Basic combined Programming language.
  - 1967, Martin Richards
* Third language :- B language
  - 1970, Ken thompson.
* Fourth language :- C language
  - 1972 Dennis Ritchie

latest version of C :- C17
- documentation 2017
- published 19 Feb 2018

& Reason behind the development of C ? B language

→ Initialy the idea was to make UNIX portable.

* IMPORTANCE OF C LANGUAGE

① efficient & fast

- debugging
- error can be corrected with ease

(4) Ability to extend ubsy

* BASIC C PROGRAM

(1) /* This is my first code. */
include <stdio.h>
void main ()
{
printf ("Hello world");
}

Output :- Hello world → screen comes for partical of second.

(2) # include <stdio.h> → preprocessor directive.
~~standard input device~~

include<conio.h> → console, input output

start ← void main ()
{
printf ("Hello world");
hold screen until anykey ← getch (); }
Output :- Hello world → screen hold

Q. What will happen if void main is absent in prog. ?

Prog will compile but it will not run
↳ error - main function is not defined

Q. WAP for calculation SI?
write a program

include <stdio.h>
include <conio.h>
void main ()
{
float P=100, R=10, T=2, SI;
SI = (P×R×T)/100;
printf ("\n simple intrest is: %f", SI);
getch ();
}

Q. WAP for addition of two no.s ?

include <stdio.h>
int main ()
{
int a, b, sum;
printf ("Enter the values of 2 integer");
scanf ("%d %d", &a &b);
sum = a+b;
printf ("%d + %d = %d", a, b, sum);
return = 0; }

Output

3 + 2 = 5

STRUCTURE OF C

```
#include <stdio.h>        → header file
#include <conio.h>
void main()               → it will not return anything
{
 statement  ← code
}
```

directly compile = Alt+F9
to save default file in .c
run → ctrl+F9

```
getch();
```

* for Addition :-

```
#include <stdio.h>
#include <conio.h>
void main()
{ int a, b, sum;
printf("Enter the two integers");
scanf("%d %d", &a, &b);
sum = a+b;
printf("sum = %d", sum);
getch();
}
```

* don't use int in division use float

* for division :-

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



```
float a, b, div;
printf("Enter the two integers");
scanf("%f, %f", &a, &b);
Div = a/b;
printf("Div = %f", Div);
getch();
}
```

for area of rectangle :-

```
#include <stdio.h>
#include <conio.h>
void main()
{ int l, b, area;
printf("Enter the value of length");
scanf("%d", &l);
printf("Enter the value of breath");
scanf("%d", &b);
area = l*b;
printf("area of rectangle")
```

for area of Square

```
{ int s, area;
printf("Enter the value of side");
scanf("%d", &s);
area = s*s;
printf("Area of Square = %d", area);
getch();
```

Date 1/1
Page 07

* Area of triangle

Triangle = $\frac{1}{2} \times b \times h$

$= (b \times h) / 2$

```
#include <stdio.h>
#include <conio.h>
void main()
{ float int a, b, h area;
printf("Enter the value of breath");
scanf("%of", &b);
printf("Enter the value of height");
scanf("%of", &h);
area = (b*h)/2;
printf("Area of triangle = %of", area);
getch();
}
```

* Diagonal of Rectangle

$d = \sqrt{(l)^2 + (b)^2}$

$d = \sqrt{l \times l + b \times b}$

$l \times l + b \times b$

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int l, b, t, d;
d = Sqrt((l*l)+(b*b));
or
t = l*l + b*b.
d = Sqrt(t);
```

① $s = ut + \frac{1}{2}at^2$

② $v^2 = u^2 + 2as$

$v = u + at$

③ $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

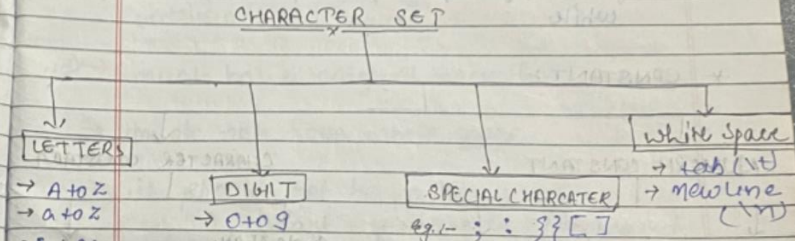
↳ pow

factorial, Peterson's no., palindrum, Reverse, Armstrong

Date 04/10/21
Page 08

* CHARACTER SYNTAX SET

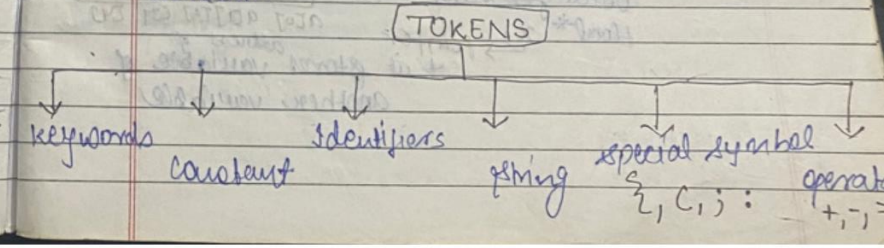
→ character set is collectively used to write the program by forming word, letter, number etc.



* TOKENS

→ Smallest individual unit of c programming language is called token.

→ Token is a source program text that the compiler does not break down into component elements.



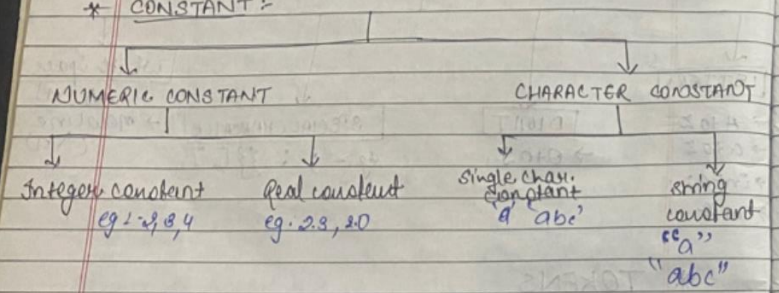
* KEYWORDS :-

Keywords have fix meaning and this meaning cannot be change, during the execution of program. Or,

Reserve words are called keywords. total no. of keywords = 32

Eg :- int, float, auto, enum, break, switch, for, while.

* CONSTANT :-



* IDENTIFIERS :-

name of variable is called identifier. (function / variable / array)

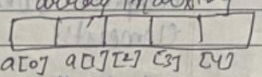
Eg :- int a ; → this is identifier

int a[10] → array

void get data ( );

float* x → pointer

* it stores address of another variable



* Rules for identifiers.

→ first character must be alphabet or underscore
int _a ✓, int a ✓, int 1ax

→ must consist of only letters, digits or underscore.

→ must have only 31 character.

→ must not have white space.

→ it should not be a keyword.

for (I ; ;) → I: start; end; increment/decrement

{ printf ("Hello");

Ex :- for (i=5 , i<5 ; i++)

{ printf ("\n H");

}

* string :-

collection of character is called string.

* no's swap a=10 b=20 → a=20 b=10

```
t = a;
a = b;
b = t
```

if problm, t=0

```
void main
int a, b, t;
&can print(a,b)
int a,b,t=0;
printf("enter the value of a");
scanf("%d", &a);
printf("enter the value of b");
scanf("%d", &b);
t = a;
a = b;
b = t;
printf("\n a=%d", a);
printf("\n b=%d", b);
getch();
clrscr();
```

05/10/21

TYPES OF DATA

- Primitive or Primary → int, float (Ex: MCT)
- derived (which is derive from primary) array, pointer
- user defined function structure union

Q. Is void main is user defined data?
Yes, but it is proto type also.

Type	Bits(size)	Bytes	Range
char or signed char	8 → 8/8 →	1	-128 to 127
char	8	→ 1	0 to 255
int or signed int	16	→ 2	-32768 to 32767
unsigned int	16	2	~~0 to 65535~~
long int (signed)	32	4	-2147,483,648 to 2147,483,647
unsigned long int	32	4	0 to 4294,967,?
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+30
long double	80	10	3.4E+4932 to 1.1E+4932

* smallest memory allocation in computer is nibble.
4 nibble = 1 bits
8 BITS = 1 Byte

BITWISE OPERATOR

- ADD AND — &
- OR — |
- Exclusive OR — ^
- Left Shift — <<
- Right Shift — >>

Date / / Page 13

* AND OPERATION

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

2^3 2^2 2^1 2^0
1 1 0 0
8 4 2 1

1 1 0 0 - 12
0 1 0 1 - 5
0 1 0 0 = 4
(12 & 5 = 4)

(both have high powers)

```
int a, b, c;
a = 12;
b = 5;
c = a & b;
printf("And operation of 12 and 5 will be", c);
```

* OR OPERATION

A	B	A | B
0	0	0
0	1	1
0	0	1
1	1	1

(in which any of them will be high, then the product will obtain)

1 1 0 0 12
0 1 0 1 5 12 | 5 = 13
1 1 0 1 13

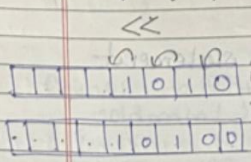
* EXCLUSIVE OR

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

1 1 0 0 12 ^ 5
0 1 0 1 = 09
1 0 0 1

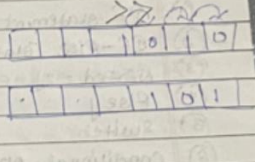
Date 06 / 10 / 21 Page 14

left shift



a = 10;
b = a<<3;
printf("%d", b);

Right shift



① void main()
{
output
void means
none
}

② int main()
{
output will
be any integer
return 0;
}

③ main()
{
output
will come
but what
don't know
}

putchar() → used to write a character

getchar() → reads a single character.

```
void main()
{
char ch;
printf("Enter a character");
ch = getchar();
printf("The character is:");
putchar(ch);
getch();
}
```

your many = char ch [ ];

Decission making and Branching

1. IF statement :-
2. IF-Else statement
3. Nested - If - else
4. Else if
5. Switch
6. Conditional operation
7. GoTo statement

} if statement

08/10/21

(I) SIMPLE IF

Syntax :-

```
if (condition)
{
  Block statement
}
```

Q WAP to print Hello if a=10?

```
void main ()
{
int a=10;
  if (a==10)
  { printf ("Hello");
  }
  getch ();
}
```

(II) IF-else statement

```
if (condition)
{
  Block sta
}
```



```
else
{
}
```

Q WAP to Print Hello if a=10 else print Hi

```
void main ()
{
  int a=20;
  if (a==10)
  {
  printf ("Hello");
  }
  else
  {
    printf ("Hi");
  }
  getch ()
}
```

Q. WAP to check whether the no is -ve or +ve
Q. WAP to check no. is even or odd
Q. WAP to find greatest among 3 no's.

```
void main ()
{ int n
  printf (" enter the number of n: ");
  scanf ("%d", &n);
  if (n<0)
  { printf (" no. is negative");
  }
```

Else

```
    }
    printf("no. is positive");
    }
    getch();
    }
```

(2) `if (n%2==0)`

```
    {
    }
else
    {
    }
```

Q. WAP to check the given sides are a of a triangle?

```
void main()
{ int a,b,c
printf("enter the value of a:");
scanf("%d",&a);
printf("enter the value of b:");
scanf("%d",&b);
printf("enter the value of c:");
scanf("%d,&c);
if((a+b)>c)&&((b+c)>a)&&((a+c)>b)
{
printf("given side are triangle")
}
else
{ printf("given side are not triangle")
}
getch();
```


Q. WAP to check, Bill > 2000, discount = 5%
else, no discount.
Bill = 2500
Dist. = 5%.
Payment = Bill - discount amount
discount amount = Bill × 0.05

```
void main()
{ float Bill, discountamount, pay};
printf("enter the Bill amount");
scanf("%f",&Bill,
if (Bill > 2000)
{
discountamount = Bill * 0.05;
pay = Bill - discount amount;
printf("please pay = %f", pay);
}
else
{ printf("please pay = %f", Bill);
}
getch()
}
```

09 | 10 | 21

NESTED EFFECTS If-else

if-else

In nested effects we write completely if else block in if block, in else block.

syntax

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

```
else {
    if (con)
    { }
  else
    { }
```

Q. WAP to find the year is leap or not.

```
void main()
{ int year;
  printf("enter a year");
  scanf("%d", &year);
    if (year % 100 == 0)
    {
      if (year % 400 == 0)
    {
    printf("It is a leap year");
    }
  else
    {
      printf("Not a leap year");
    }
else {
      if (year % 4 == 0)
    { printf("leap")
    }
  else
    { printf("not a leapyear")
    }
getch();
    }
```


Q. WAP to find greatest among 3 no.'s

18/10/21

* LADDER - IF ELSE

Syntax

```
if (condition)
  { statement
    }
else if (condition)
    { statement }
else {
    }
```

Q. WAP to calculate electricity bill. Read the starting and ending meter. The charges are as follows.

No. of unit's consumed	Rates in (Rs)
200 - 500	3.50
100 - 200	2.50
less than 100	1.50

↳ initial reading
final reading
unit consumed
Amount Bill

```
void main ( )
{
float ir, fr, u, Bill = 0
printf ("enter the initial reading");
scanf ("%f", &ir);
printf ("enter the final reading");
scanf ("%f", &fr);
u = fr - ir;
```

Page 21

```
if ((u >= 200) && (u <= 500))
{
  Bill = u * 3.50;
  printf ("please pay = %f", Bill);
}
else if ((u >= 100) && (u < 200))
{
  Bill = u * 2.50;
  printf ("Please pay = %f", Bill);
}
else if (u < 100)
  {
        }
else {
      printf ("Not valid");
     }
```

BREAK AND CONTINUE

continue = loop will continue to iterate

break = it will break and it will get terminate from the loop

```
int i;
for (i = 1; i < 10; i++)        It will get
{                                out of our
  if (i == 5)                    loop
    break;
  printf ("This is break statement's for = %d", i);
}
for (i = 1, i < 10  i++)
{
  if (i == 5)
  continue;
  print ("This statement continues for = %d", i);
}
```

Page 22

dry run

The break statement is for 1, 2, 3, 4

Break will get when it will be equal to 5

continue will get will be same, but it will leave 5: for 1, 2, 3, 4, 6, 7, 8, 9

© both switch and loop that transfer control of the program to the another part of program

only in loop

20.10.21

SWITCH (EXPRESSION)

```
{ CASE 1 :
        break;
  CASE 2 :
        break;
  DEFAULT :
        break;
}
```

The switch statement ~~taste~~ tests the value of a given variable or expression against a list of case values and when a match is ~~fourned~~ found a block of statement associated with that case is executed.

Q WAP to write the as :-

1 → sunday

2 → monday

⋮

7 → saturday

```
void main()
{
 int a, ch;
 printf("Enter the value");
 scanf("%d", &a);
 ch = a;
 switch (ch)
 {
  case 1:
    printf("In sunday");
    break;
  case 7:
    printf("In saturday");
    break;
  default:
    printf("Not valid")
    break;
  }
 getch()
  }
```

8. WAP for :-
1- add, 2- sub, 3- Multiply, 4- divide

```
void main()
{
 int a,b,c=0,ch;

printf("Enter 1st no.");
scanf("%d", &a);
```



```
printf("Enter 2nd no.");
scanf("%d", &b);
printf("enter your choice");
scanf("%d", &ch);
switch(ch)
{
case 1:
  c=a+b;
printf("The sum is=%d", c);
break;
case 2:
  c= a-b;
printf("substraction= %d", c);
break;
case 3:
  c=a*b;
printf("Multiply=%d", c);
break;
case 4:
  c= a/b;
printf("The division of a and b is =%d", c);
break;
default;
  printf("Invalid choice");
break;
}
getch();
}
```

H.W. Q. WAP while using switch()
1 → perimeter of Δ
2 → area of Δ
3 → perimeter of rectangle
4 → area of square
5 → perimeter of square
6 → Area of square
7 → T.S.A of cube
8 → T.S.A of cuboid

* GOTO STATEMENT :- / (Jump statement)
→ forward jumps
→ backward jump

syntax:-

```
goto label;
  ====
label:
  statement;
  statement;
```

forward jump

→ It will get out of all loop

```
label:
  ====   (execute) (it will execute this)
  ≡
goto label;
```

(then) backward jump

LOOPS

1. while loop
2. DO-while
3. for loop

while	DO-while

while (condition)
{
statement
}
	do
	{
	statement
	}
	while(condition)

eg:- while (n>2)
{
print ("Hello")
}
	do
	{ print ("Hello")
	} while (n>2)

for (initial, final, step value)

```
for (i=0; i<3; i++)
{
print ("Hello")
}
```

Define loop :- A loop is defined as a block of statement which are repeatedly executed for a certain no. of times

Difference between while and dowhile loop

while

while loop	do-while loop
1) while (condition) { statement }	do { statement } while (condition)
2) condition is check in beginning	condition checked at last
3) It is known as entry-control loop	It is known as exit control loop
4) If condition is false statement will not execute	statement will execute at least once
5) while (n>2) { print("Hello"); }	do { print("Hello") } while (n>2)

Q. WAP to print "Hello" for 5 times.

```
while (n<=5)
{
print ("Hello");
n++;
}
```

WAP to print 1+2+3+4+5 = ? 15

```
int n=1, sum=0;
while (n<=5)
{
sum = sum+n;
printf("sum %d", sum);
n++;
}
```


dry run

sum = 0+1 = 1
sum = 1+2 = 3
sum = 3+3 = 6
sum = 6+4 = 10
sum = 10+5 = 15

(7) WAP to print 1+1/2+1/3+1/4+1/5 ... 1/n

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

22/10/21

Q. WAP to find

```
void main()
{ int n, i=1;
float sum=0;
printf("Enter the value of n");
scanf("%d", &n);
while (i<=n)
{
sum = sum + 1/i;
i++;
}
printf("sum = %f", sum);
getch();
}
```

Dry Run.

n	i	sum
5	1	1.5

i=2

Enter the value of n: 5

sum = sum + 1/i

sum = 1 + 1/5

sum

Q. WAP to find sum of natural no.'s
i.e., 1+2+3+4+.....+n

```
void main()
{ int n, i=0, sum=0;
  printf("enter the value of n");
  scanf("%d", &n);
  while (i<=n)
  {
    sum = sum + i;
    i++;
  }
  printf("sum = %d", sum);
  getch();
}
```

Dry Run

1+2+3+4+5=15

Dry run

n	i	sum
5	1	0

Enter the value of n 5

i=1 sum = sum + i = 0+1

sum = 1+2

i=2 sum = 2+3

i=3 sum = 6+4

i=4 sum = 10+5

i=5 sum = 15

Q. WAP to find odd no's upto 50

```
void main()
{ int i=1
  while (i<=50)
  {
    printf("odd = %d", i);
    i+2;
  }
  getch()
  }
```

Q. WAP to sum of given number i.e., 153 = 1+5+3 = 9

```
void main()
{
 int n, t, sum=0;
 printf("enter the no.'s")
 scanf("%d", &n);
 while (n>0);
 {
    t = n % 10;
    sum = sum + t;
     n = n/10;
 }
printf("sum of the given no. is = %d", sum);
getch();
}
```

Q. WAP to reverse a number

```
int
void main()
{
int a, rem, reverse = 0;
printf("Enter the no. you want to reverse: ");
scanf("%d", &a);
while(a!=0)
{
  rem = a%10;
  reverse = (reverse*10) + rem;
  a = a/10;
}
printf("reverse no. is: %d", reverse);
getch();
}
```

Dry Run

a	rem	reverse
121		
12	1	1
1	2	12
0	1	121

① 121/10 = 1
rem = 1*10+1 = 11
n = 12

② a = 2*10+1 = 21

Q. WAP to for palindrom no.

```
void main()
{ int n, t=0, sum=0, a;
printf("Enter the no.");
scanf("%d", &n);
a=n;
```



```
while(n>0);
{
t = n%10;
sum = (sum*10) + t;
n = n/10;
}
if(sum==a)
{
printf("P");
}
else
{
printf("NO.P");
}
getch();
}
```

25 | 10 | 21

Q. WAP for fibonacci series upto 12

```
a=0, b=1, c, i=1
printf("%d\n", a);
printf("%d\n", b);
while(i<=10)
{
  c = a+b;
  a = b
  b = c
  printf("%d\n", c);
  i++
}
getch();
}
```

Q. Write a program for armstrong no.:-

```
void main()
{
int n,t,sum=0,a;
printf("Enter the no.'s");
scanf("%d",&n);
a=n;
while(n>0)
{
  t=n%10
  sum=sum+(t*t*t);
  n=n/10;
}
if(sum==a)
  printf("Armstrong");
else
  printf("Not Armstrong");
getch();
}
```

26/10/21

* For loop

Syntax:-

```
for(initial value; final value; step value)
{
    statement;
}
```

Eg:-

```
for(i=0; i<=5; i++)
{
printf("Hello");
}
```

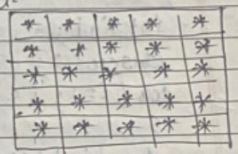

`int i=0;`	`int i=0;`	`int i;`
`for(;i<=5;i++)`	`for(;i<5;)`	`for(i=0;i<5;)`
`{ ; }`	`{ i++; }`	`{ i++; }`

Q. WAP to find factorial of 5

```
int i,fact=1;
printf("finding factorial");
for(i=1;i<=5;i++)
{
  fact*i;
}
printf("factorial is %d",float);
```

27/10/21

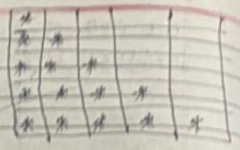
Q. WAP to draw pattern given:-



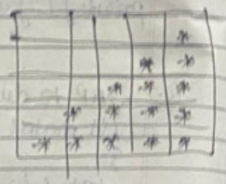
```
void main()
{
int i,j;
clrscr();
for(i=1;i<=5;i++)
{
for(j=1;j<=5;j++)
{
  print("*");
}
printf("\n");
}
getch();
}
```

Q. WAP to draw pattern given

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

```
void main()
{
 int i,j;
 clrscr();
for (i=1; i<=5; i++)
{
for (j=1; j<=i; i++)
{
  printf ("*");
}
 printf ("\n");
}
getch();
```

Q. WAP to draw pattern given:-

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

```
 void main()
 {
  int i,j;
  clrscr();
  for (i=1; i<=5, i++)
  {
  for (j=1; j<=5, j++)
  {
   if ((i+j)>=6)
  { printf ("*");
  }
else
  { printf (" ");
  }
  printf ("\n")
}
getch()
```


ARRAY

It is a collection of similar data type.
There are two types of array :-
(i) one-dimensional array
(ii) two dimensional array

One dimensional array:- it is also called linear array.

Syntax:-
data-type array-name [size]; [subscript]
Eg:- int a[10];

a[0] a[1] ... a[9]

Two dimensional array:-
Syntax:-
data-type array-name [size] [size];
int a [10] [10];

why array?
advantage of array :-
→ with the help of array we can store large volume of similar data and through these data it will be very easier to perform various operation such as
(i) traversing :- visiting each element atleast once
(ii) search :- finding the location of element
(iii) insertion :- adding a new element to the list

④ Sorting :- arranging the data in ascending order.

⑤ deletion :- removing an element from the list

⑥ Merging :- combining two list in a single list

Disadvantage of array :-

→ it is relatively expensive to insert and delete in an array.

→ since an array usually occupies a block of memory space, one cannot simply double or triple the size of an array.

*→ array is also called dense list and said to be static data structure.

Q. WAP to add 10 numbers.

```
void main()
{
int a[10];
int i, sum=0;
printf("enter the 10 no.'s");
for(i=0; i<=9; i++)
  {
  scanf("%d", &a[i]);
  }
for(i=0; i<=9; i++)
```



```
  {
  sum = sum + a[i];
  }
printf("sum=%d", sum);
getch();
}
```

Q. WAP to enter 10 no.'s & display even and odd.

```
void main()
{
int a[10];
int i, sum=0;
printf("enter 10 no.'s");
for(i=0, i<=9, i++)
  {
  scanf("%d", &a[i]);
  }
for(i=0; i<=9; i++)
if(a[i]%2==0)
printf("%d is even no.", a[i]);
else
printf("%d is odd no.", a[i]);
}
getch();
}
```

Q WAP to enter 10 no. and display the longest among them.

```
void main()
```

Post-increment

```
int n=1;
while (n<10)
{
n++;
printf("value of n is =%d", n);
}
```

Pre-increment

```
int n=1;
while (n<10)
{
++n;
printf("value of an is =%d", n);
}
```

* Two-Dimensional Array

```
void main()
{
int a[3][3], i, j;
clrscr();
printf("Enter the element of Matrix a: ");
for(i=0; i<=2; i++)
{
for(j=0; j<=2; j++)
{
scanf("%d", &a[i][j]);
}
}
printf("matrix is: \n");
for(i=0; i<=2; i++)
{
for(j=0; j<=2; j++)
{
printf("%d", a[i][j]);
printf("\n");
printf("\n");
getch();
}
```

* TRANSPOSE of Matrix

$a_{ij} \rightarrow a_{ji}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

↑ Two dimensional array

```
printf("Transpose matrix is: \n");
for(i=0; i<=2; i++)
{
for(j=0; j<=2; j++)
{
printf("%d", a[j][i]);
}
printf("\n");
}
getch();
}
```

* Adding two matrix c[i][j] = a[i][j] + b[i][j];

HW

```
{
int a[2][2], b[2][2], i, j, c;
clrscr();
printf("Enter the element of matrix a: ");
scanf("%d", &x);
```

```
scanf("%d%d", &m, &n);
printf("enter the elements of first matrix\n");

for(c=0; c<m; c++)
 for(d=0; d<n; d++)
   scanf("%d", &first[c][d]);

printf("enter the elements of second matrix\n");
printf("sum of entered matrices:-\n");
printf("Enter the elements of second matrix\n");

for(c=0; c<m; c++)
 for(d=0; d<n; d++) {
 scanf("%d", &second[c][d]);
printf("sum of entered matrices:-\n");

for(c=0; c<m; c++)
 {
  for(d=0; d<n; d++)
{
  sum[c][d] = first[c][d] + second[c][d];
printf("%d\t", sum[c][d]);
}
printf("\n");
}
return 0;
}
```

* Substraction of two matrices:-

```
{
```



```
difference[c]
[d] = first[c][d] - second[c]
[d];
 printf("%d\t", difference[c][d]);
}
printf("\n");
}
return 0;
}
```

* division of two matrices:-

```
c[i][j] = (a[i][j])/(b[i][j]);
return;
}
divmatrix(a, b);
printf("\nDivided matrix :\n");
for(i=0; j<3; i++)
 { for(j=0; j=3; j++)
    printf("%.7f, ", c[i][j]);
    printf("\n");
}
```

30/10/21

* character array:-
collection of characters are called character array.

Syntax:-
character_datatype [20];
eg. char name [20]

Each element of array placed in a definite

memory space and each element can be access separately.

The array element should end with null character as a reference for termination of character array.

The null character will be added automatically by C compiler provided there is enough space to accomodate the character

Charname[6] ← | R | O | H | A | N | \0 | → null character

Charname[5] = ROHAN → error

Q. WAP to find the length of a string without using string function?

```
void main()
{
char s1[15];
int l=0,i;
clrscr();
printf("enter a string");
scanf("%s",s1);
for(i=0;s1[i]!='\0';i++)
{
l=l+1;
}
printf("\n length of a string=%d",l);
getch();
}
```


Q. compare two strings without using string function?

```
void main()
{
char s1[15];
char s2[15];
int i,k=0;
clrscr();
printf("enter the first string\n");
scanf("%s",s);
printf("enter the second string\n");
scanf("%s",s2);
for(i=0;(s1[i]!='\0'||s2[i]!='\0');i++)
{
if(s1[i]!=s2[i])
{
k=1;
break;
}
}
if(k==0)
{
printf("strings are equal");
}
else
{
printf("strings are not equal");
}
getch();
}
```

output :-

enter the first string
Rohan
enter the second string
Rim
strings are not equal

Q. WAP to copy a string without using string function.

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char s1[15];
  char s2[15];
  int i;
  clrscr();
  printf("Enter a string");
  scanf("%s", s1);
  for(i=0; s1[i]!='\0'; i++)
  {
    s2[i]=s1[i];
  }
  s2='\0';
  printf("\n string s2=%s", s2);
  getch();
}
```

* String Handling functions:-

Function Name	Action
strcat()	concatenates 2 strings
strcmp()	compares 2 strings
strcpy()	copies a string
strlin()	finds the length of a string
strrev()	reverse a string
strupr()	converts in uppercase

strlwr() converts a string in lowercase

```
#include <stdio.h>
#include <conio.h>
void main()
{
  char s1[15];
  char s2[15];
  char s3[31];
  int i;
  clrscr();
  printf("Enter a string");
  scanf("%s", s1);
  printf("Enter 2nd string");
  scanf("%s", s2);
  s3 = strcat(s1, s2);
  printf("%s", s3);
  getch();
  strcat(s1, s2);
  printf("%s", s1);
  getch();
}
```

* Function :-

A complex problem may be decompose into a smaller or easily manageable parts or module, called function.

Reverse :-

Function is the block of statements which can call by another function.

Reverse :-

```
Void main ()
{
  char s1[15];
  char s2[15];
  clrscr ();
  printf ("enter a string");
  scanf ("%s", s1);
  s2 = strrev (s1);
  printf ("reversed string = %s" s2);
  getch ();
}
```

Function is very useful to read, write, debug and modify complex program.

* Advantages of using function are :-

- Easy to write a correct small function.
- easy to read, write & debug.
- easier to maintain or modify

* Types of Function :-

1) User - Defined function
2) Built - in function. (Ex:- printf, scan() etc.)

Q: Is void main a built in function or user defined function?

↳ void main() is a user-defined function

* A function is a set of statement that takes input, do some specific computation and gives output.

but main is a predifened signature.

* defining a function

A function definition has a name, a parenthesis pair containing zero or more parameters and a body.

function type function-name (datatype argument1, datatype argument2)

```
{
  ==== Body
  return (something)
}
```

```
int add (int, int);
Void main ()
{
int x=1, y=2, z;
z = add (x, y);
printf ("z= %d", z);
}
int add (int a, int b)
{
 return (a+b);
}
```

(A) Top- Down Approach :-

* In top- down approach main function is written first in which function is called and function

is defined outside the main function.

- In this approach, a function prototype must be declared before the main function.

(B) Function Prototype :-

- Describe how the function is called.
- Tells everything you need to know to make a function call.
- Formal parameter is a place holder to stand for actual parameter.
- Function prototype terminates with semicolon (;).

```
#include <stdio.h>
#include <conio.h>
void sum (void);
int x, y;
void main ()
{
printf ("Enter two nos");
scanf ("%d%d", &x, &y);
sum ();
getch ();
}
void sum (void)
{
int z;
z = x+y;
printf ("sum = %d", z);
}
```


(C) BOTTOM-UP APPROACH :-

In bottom-up approach function is defined first, after that main program is written.
There is no need to declare function prototype.

* categories of function :-

(i) Function taking no arguments and not returning value.

(ii) Function taking arguments and not returning value.

(iii) Function taking no arguments but returning a value.

(iv) Function taking argument and returning a value.

* Formal argument :-
Any variables declared in the body of a function is said to be local to that function.

```
void square (int a, int b)
{
=========
}
```

* Function taking argument and not returning value

```
#include <stdio.h>
#include <conio.h>
void sum (int x, int y);
void main()
{
int x, y;          → main function has two variable x, y
clrscr();
printf("enter two numbers");
scanf("%d %d", &x, &y);
sum(x,y);
getch();
}
void sum (int a, int b)  → passing two argument
{
int c;
c=a+b;
printf("sum=%d", c);
}
```

* Function taking no argument but returning a value.

```
# include <stdio.h>
# include <conio.h>
int sum (void);                    ] function declaration
int x,y;
void main()
{
int x;
clrscr();  → clear screen library func.
printf("Enter two no.");
scanf("%d %d", &x, &y);
x = sum();   → variable
printf("sum = %d", z);
getch();
}
int sum (void)                     ] function defination
{
int c;
c = x+y;
return (c);
}
```

function calling

* function taking argument and returning a value.

```
# include <stdio.h>
# include <conio.h>
int sum (int x, int y);
void main()
{
int x,y,z;
clrscr();
printf("Enter two no.");
scanf("%d %d", &x, &y);
z = sum(x,y);
printf("Sum = %d", z);
getch();
}
int sum (int a, int b)  → take input
{
int c;
c = a+b;
return (c);
}
```


* ASSIGNMENT

(i) WAP to find the factorial of a no. using function

(ii) WAP to reverse a digit using function

1→

```
# including <stdio.h>
# including <conio.h>
int fact (int a);
void main()
{
int a, b;
printf("Enter a no.");
scanf("%d", &a);
```

```
b=fact(a);
printf("The factorial of x: %d", b);
getch();
}
int fact(int x)
{
int i, c=1;
for(i=1, i<=x; i++)
{
c=c*i;
}
return(c);
}
```

(2)

```
#include <stdio.h>
#include <conio.h>
int reverse(int);
void main()
{
int a, b=0
printf("Enter a number to reverse the no.");
scanf("%d", &a);
b=reverse(a);
printf("The reverse of the no. is %d", b);
getch();
}
int rev(int c)
{
int d, i=0
while(c!=0)
{
```

```
d=c%10;
i=i*10+d;
c=c/10;
}
return(i);
}
```

16/11/21

* Repetation :-

→ By using for, while & do-while loop (Iterative method)
→ By using function (function calling itself)

* Recursion :-

Sometimes a function is required to be called in the body of the same function in order to incourage repetative operations a no. of items such functions are called recursive functions.

Syntax :-

```
void main()
{
statement;
m1();
}
void m1()
{
m1(); called it one more
}
```

* Types of Recursion :-

(I) Direct Recursion
(II) Indirect Recursion
(III) Tail Recursion
(IV) Non-Tail Recursion / Head Recursion
(V) Linear Recursion
(VI) Tree Recursion

(I) Direct Recursion :-

When a function call itself within the same function repeatedly

it is called the direct recursion.

Syntax:-

```
fun1()
{
//code
fun1()
//code
}
```

code:-

```
#include <stdio.h>
#include <conio.h>
int factorial(int n);
void main()
{
int n, f;
clrscr();
printf("Enter the no.");
scanf("%d", &n);
f = factorial(n);
printf("factorial: %d", f);
getch();
}
int factorial(int a)
{
int fact;
if(a==1)
{
return 1;
}
else
```

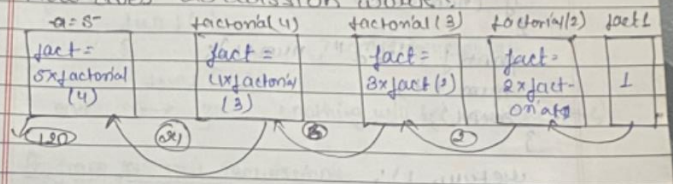


```
fact = a * factorial(a-1);
}
return(fact);
}
```

17/11/21

Dry Run:-

How does recursion works?



* Indirect Recursion:-

When a function is mutually called by another function in a circular manner, the function is called Indirect recursion method.

Syntax:-

```
fun1()
{
fun2()
}
fun2()
{
fun1()
}
```

Q. WAP for odd-even code using recursion:-

```
# include <stdio.h>
void odd();
void even();
int num=1;
void odd()
{
 if (num<=10)
 {
 printf("%d", num+1);
 num++;
 even();
 }
 return 1;
}
void even()
{
 if (num <=10)
 {
 printf("%d", num-1);
 num++;
 odd();
 }
 return;
}
int main()
{
 odd();
 return 0;
}
```

Output:-
2 1 4 3 6 5 8 7 10 9

① does not perform any computation after recursive call.
or, recursive call is the last thing done by the function.

* TAIL RECURSION:-
A recursive function is used to call tail recursive if function meet mixed recursive calling they and that the recursive call is the last statement execute by the function.
after that there is no function or statement is left to call the recursive function.

Syntax:-

```
fun1()
{                 { write here only }
  fun2()
end → }           (nothing will be written here)
```

Program for Tail Recursion:

```
# include<stdio.h>
void fun1 (int num)
{
 if (num==0)
  return;
 else
 printf ("\n Number is %d", num);
 return fun1(num-1);
}
int main()
{
 fun1(7);
 return 0;
}
```

Output:-
7 6 5 4 3 2 1

dry run:-
~~7-1=6~~
7==0
7-1=6
6-1 5
5-1 4
4-1 3
3-1 2
2-1 1

① recursion is not the last thing done by the function.
② there will be something left to evaluate.

Date / / Page 61

* HEAD RECURSION

A function is called head recursive if a function makes a recursive call itself, the recursive call will be the first statement in the function.
It means there should be no statement or operation is called before the recursive call.
The head recursion does not perform any operation at the time of recursive calling. Instead all operations are done at the return time.

Program:-

```
#include <stdio.h>
void head_fun (int num)
{
 if (num>0)
  {
   head_fun (num-1);
   printf ("%d", num);
  }
}
int main()
{
int a=5;
printf ("Non Tail Expression");
head_fun(a);
return 0;
}
```

* memory allocation linearly
* size also linearly

Date 23/11/21 Page 62

* Linear Recursion:-

A function is called the linear recursive if the function makes the single call to itself at each time the function runs which grows linear in proportion to the size of the problem.

Program:-

```
#include <stdio.h>
#define NUM 7
int rec_num (int *arr, int n)
{
 if (n==1)
  {
   return arr[0];
  }
return Max_num(rec_num(arr, n-1), arr[n-1]);
}
int Max_num (int n, int m)
{
 if (n>m)
  return n;
 return m;
}
int main()
{
int arr[NUM]={4,8,23,19,19,5,35,2};
```

```
int max = rec_num (arr, NUM);
printf ("Maximum no. %d", max);
}
```

* TREE RECURSION:-

A function is called tree recursion in which the function makes more than one call to itself with in the recursive function

Program:-

```
#include <stdio.h>
int fibo_num (int num)
{
 if (num<=1)
 return num;
 return fibo_num (num-1) + fibo_num (num-2);
}
void main ()
{
 int num=7;
 printf ("use of tree");
 printf ("fibonacci series is:%d", fibo_num(7));
}
```


* STRUCTURE

In C, collection of heterogeneous data type can be grouped to form a structure. When this is done, the entire collection can be referred to by a structure name. The individual component which are called fields or members can be accessed and processed separately.

Syntax:

```
Struct user-defined-name:        struct student
{                                {
char name[20];                    member 1;
int age;                          member 2;
int R.No;                        };
};
```

Difference between structure and array

→ All the element of an array have the same type, where as in a structure the components of fields can have different data type

→ Component of an array is referred to by its position, where as each components of a structure has a unique name.

* Structure and array have a similarity

that is i.e, both must be defined within a infinite number of components.

27/11/19

Q WAP using structure to display the detail of student. (for 5 student)

```
#include <stdio.h>
#include <conio.h>
struct student
{
char num[20];
int roll;
int marks;
};
void main()
{
struct student std;
clrscr();
printf("Enter name");
scanf("%s", std.num);
printf("Enter roll");
scanf("%d", &std.roll);
printf("Enter marks =");
scanf("%d", &std.marks);
printf("........");  (anything to define)
printf("Student details");
printf("........");
printf("Name = %s \n roll = %d \n Marks = %d",
        std.num, std.roll, std.marks);
getch();
}
```


Array within structure :-

WAP using structure to display the detail of student (for more student)

```
#include <stdio.h>
#include <conio.h>
struct student
{
char num[20][10];
int roll[10];
int marks[10];
};
void main()
{
int i;
struct student std;
clrscr();
for(i=0; i<=9; i++)
{
printf("Enter name");
scanf("%s", std.num[i]);
printf("Enter roll");
scanf("%d", &std.roll[i]);
printf("Enter marks");
scanf("%d", &std.marks[i]);
}
for(i=0, i<=9, i++)
{
printf("Student detail : %d", i+1);
printf("Name = %s \n Roll = %d \n marks = %d",
  std.num[i], std.roll[i], std.marks[i]);
```

→ * union is a special type of data type that allow to store different types of a datatype in the same memory location.
→ one member only contain a value at a time.

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

Q. difference between structure and union (5)

structure is a container defined in C to store data

29 | 11 | 21

* Union:-

→ It is similar to structure data type with a difference in a way the data is stored and retrieved.

→ The union stores values of different type in a single location.

Syntax:-

```
union user-defined-stname
{
    member1;
    member 2;     datatype
                  take variable
                  1 variable at a time
};
```

Program:-

```
#include <stdio.h>
#include <conio.h>
union date
{
    int day;
    int month;
```



```
    int year;
};
void main ()
{
    union date dt1, dt2, dt3;
    clrscr();
    printf ("Enter day = ");
    scanf ("%d", &dt1.day);
    printf (" Enter month = ");
    scanf ("%d", &dt2.month);
    printf ("Enter year = ");
    scanf ("%d", &dt3.year);
    printf (".........");
    printf ("Day= %d\n Month = %d\n year= %d",
            dt1.day, dt2.month, dt3.year);
    getch();
}
```

Memory allocation

Structure st1	Union std.
{	{
int roll no; -2	int Roll no; -2
int admission no; -2	int admission no; -2
char name[10]; -10	char name [10]; - 10
};	};
=14	10
(the total no. of blocks and their sum will be the memory allocation)	(the biggest block no. will be the memory allocation)

(reverse polish notation)

* SDS matrix :-

STACK

→ stack is a linear data structure that follows a particular order in which the operations are performed.

The order may be LIFO (Last in First out) or FILO (first in last out)

Basic operations of stack:-

1. push()
2. pop()
3. peek or top()
4. is empty()
5. is full()

(1) push() → adds an item in the stack. If the stack is full then it is said to be an over flow condition (over flow condition)

(2) pop() → removes an item from the stack. If the stack is empty then it is said to be an under flow condition.

(3) peek or top() → returns the top element of the stack

(4) is Empty() → Returns true if the stack is empty, else false.

(5) is full() → Returns true if stack is full else false.

* Time complexities of operation stand.

push(), pop(), is empty() and peek() all takes O(1) → (being O of one) of time.

• Application of stack :-

(i) Evaluation of arithmetic expressions
(ii) Back tracking
(iii) delimiter checking
(iv) Reverse a data
(v) processing function calls

Advantage :-

(1) Easy to implement

• Disadvantage :-

(i) it is not dynamic
(ii) it doesnot grow and shrinks depending on needs at one run time

Queue

→ Like stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is FIFO (first in first out)
→ Insertion takes place at rear end.

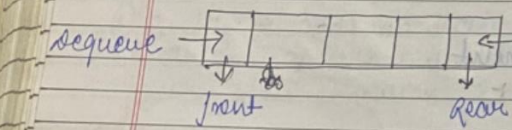
Basic operation of queue –

Enqueue :- adds an item to the queue. if queue is full then it is said to be an overflow condition.

Dequeue :- removes an item from the queue. if the queue is empty then it is said to be an underflow condition.

front :- get the front item from the queue

Rear :- get the last item from the queue.



Representation of Queue

* Application of Queue

① when a resource is shared among multiple

consumers.
Example :- CPU scheduling, disks scheduling,
② data is transferred asynchronously.
③ in operating system,
  (i) semaphores
  (ii) spooling in printer
④ in routers / switches
⑤ mailqueues

* Advantage of Queues :-

① Easy to implement

* Disadvantage :-

① static data structure
② fixed size

* Different type of Queues :-

① simple Queues
② Circular Queues
③ Priority Queues
④ Double ended Queues

① simple Queue :-

In a simple queue, insertion takes place at rear and removal occurs at the front.

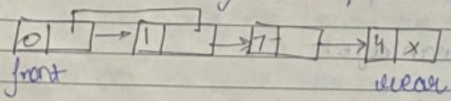
⇒ strictly follows FIFO

* Linked Queue :-

Each node of the queue consist of two parts i.e., data part and linked part.
There are two pointers maintain in the memory (front pointer and rear pointer)

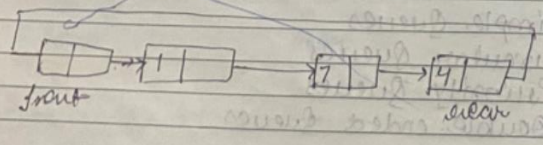
front pointer: address of the starting element of the queue.

rear pointer: contains the address of the last element of the queue.



(iv) Circular queue :-

In a circular queue, the last element point to the first element making a circular link.

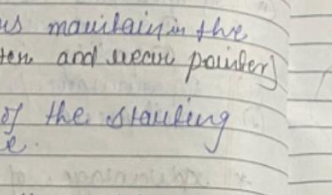


* Advantage of circular queue over a simple queue is better memory utilization.

If the last position is full and the first

position is empty we can insert an element in first position.

* Representation of circular queue



It is also known as ring buffer

* Application of Circular Queue

(i) memory management
(ii) C.P.U scheduling
(iii) traffic system

01-12-21

IMPLEMENTATION

(i) using array
(ii) using link list

Evaluation of arithmetic expressions

(i) Notation of arithmetic expression
- (i) infix notation
- (ii) prefix notation

(iii) postfix notation.

(1) infix notation:- the infix notation is a convinient way of writting an expressions in which each operator is placed between operands.

Ex:- a + b

2 + 3 ; 2 - 3 ; 2-3 * e

(2) Prefix notation:- In this notation operator is prefix to operants (operator is written ahead of operants).

Also known as polish notation.

Ex:- 2+3 = +23	(2/3) + (7/2)
2*3 = *23	/23 + /72
(2*3)+7 = +*237	+ /23/72

(iii) postfix notation:-

(i) it is also known as reverse polish notation.

(ii) In this notation operator is written after the operants.

Ex:- 2+3 = 23+ 2* 3+7 = 23*7+

2*3 = 23*

Ex:- (2/3) + (7/2)

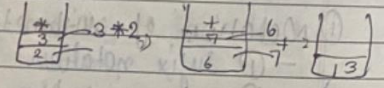
23/ + 72/

23/72/+

How it work in stack?

if (2*3)+7 = 13

⇒ 23*7 +



PUSH

Empty?

Yes → operation overflow

No ↓

POP

is full ()

Yes → pop underflow

No ↓

Code:

```
#include <stdio.h>
int MAX SIZE = 8;
int stack [8];
int top = -1;
int is Empty ()
{
if (top == -1)
return 1;
else
return 0;
}
int is full()
{
if (top == Max size)
return 1;
else
return 0;
}
int is full()
{
if (top == MAX SIZE)
return 1;
else
```

```
int pop()
{
int data;
if ( ! is Empty() )
{
data = stack [top];
top = top - 1;
return data;
}
else
{
printf ("Stack is Empty");
}
}
int push (int data)
{
if ( ! is full())
{
top = top + 1
stack[top] = data;
}
else
```

Page 17

```
printf("stack is full");
}
int main ()
{
push (3);
push (5);
push (9);
push (1);
printf ("Top element %
    d/n, " peek ());

printf ("Elements in
while ( ! is empty ())
{
int data = pop ();
printf ("%d)n ", data
}
return 0;
}
```

04-12-21

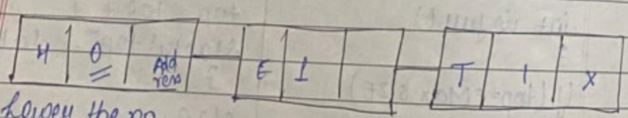
Priority Queue. :-
A priority queue is a special type of queue in which each element is associated with a priority and served according to its priority.

*→ If elements with the same priority occurs they are served ~~to~~ according to their order in the queue.

→ Insertion occurs based on the arrival of the values as removal occurs based on priority.

→ It will also not follow FIFO rule.

Info	Priority	Pointer



Lower the no.
Higher the priority

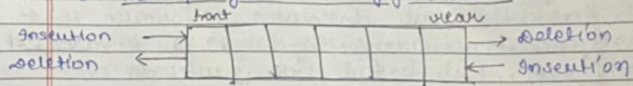
Page 18

→ when the Hie will come then the first who came it will be executed.

* De Queue :- double ended queue

→ In De queue insertion and removal of an element can be performed from either, from ~~performed~~ front or rare rear

*→ It does not follow the fifo rule.



* ALGORITHM TO INSERT AN ELEMENT IN A QUEUE

INSERTION

Step 1 :- If rear = Max - 1
write over flow
GOTO STEP 4
[END OF IF]

Step 2 :- If FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE
SET REAR = REAR + 1
[END OF IF]

Step 3 :- SET QUEUE [REAR] = NUM

Step 4 :- EXIT

DELETION

Step 1:- If FRONT=-1 or FRONT > REAR
write underflow
SET VAL = QUEUE [FRONT]
SET FRONT = FRONT + 1
[END OF IF]

Step 2:- EXIT

06-12-21

* POINTER

→ Pointer is a powerful technique to access the data by indirect reference as it holds the address of that variable where it has been stored in the memory.

POINTER DECLARATION

→ a pointer is a variable which holds the memory address of another variable.

* Advantages of Pointer:

→ It allows to pass variables, arrays, functions, strings and structures as function argument.

→ A pointer allows to return structured

variable from functions.

→ It supports dynamic allocation and deallocation of memory segments.

→ With the help of a pointer variables can be swip without physically moving them.

→ It allows to establish links between data elements or object for some complex data structure, ex:- linked list, stack, queues, recursion and graphs.

* A pointer variable consist of two parts, pointer operator, address operator.

Assignment:-

write a program to print your name without using semicolon except in clrscr() n getch().

write a program add to two no. without using preprocessor directive.

Q. write a program to print address of a variable

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

```
void main()
{
int x=5;
printf("%d is stored at address=%u", x, &x);
getch();
}
```

output:-

5 , 5 is stored at address = 217356

217356

Q WAP to illustrate the pointer use of pointer

```
#include <stdio.h>
#include <conio.h>
void main()
{
int x, *p;
clrscr();
x=8;
p=&x;
printf("%d is stored at address=%u", x, &x);
printf("\n %d is stored at address=%u", x, p);
printf("\n %d is stored at address=%u", *p, p);
printf("\n %u is stored at address=%u", p, &p);
getch();
}
```

output:-

x = 8 at 65488; p = 65490

⇒ 8 is stored at address = 65488

⇒ 8 is stored at address = 65490

⇒ 65490 ⟶ 65490

Assignment:-

①
```
#include <stdio.h>
int main()
{
if(printf("PRIYANGANN"))}}
return 0;
}
```

07-12-21

Q② WAP to add two numbers using python. (pointer)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int x, y;
int *a, *b, c;
a=&x;
b=&y;
clrscr();
printf("Enter two no.'s");
scanf("%d %d", &x, &y);
c=*a+*b;
printf("\n sum=%d", c);
getch();
}
```

dry run:-

x = 2 at 12; y = 3 at 14; a = 12 at 18; b = 14 at 20; c = 22

output:-

Enter two no's

2↵

3↵

sum = 5

* Call by Value and Call by Reference

bottom up approach

SWAPPING CALL BY VALUE

```
#include <stdio.h>
# include <conio.h>
void swap (int, int);      = prototype
int main()
{
 int a = 10;
 int b = 20;
printf ("Before swap a = %d, b= %d", a, b);
swap (a, b);        actual parameter
printf ("After swapping a = %d, b = %d", a, b )
}
void sum (int a, int b)  = formal parameter
{
 int temp; → variable
 temp = a;
 a = b;
 b = temp;
printf ("After swap a = %d, b = %d", a, b)
}
```

output :-
Before swap , a = 10 , b = 20
after swap , a = 20, b = 10
After swapping, a = 10, b = 20

SWAPPING CALL BY REFERENCE

```
# include <stdio.h>
# include <conio.h>
void swap (int*, int*);
int main()
{
 int a = 10;
 int b = 20;
printf ("Before swap a = %d, b = %d", a, b);
 swap (&a, &b);
printf ("After swapping a = %d, b = %d", a, b);
}
void swap (int *a, int *b)
{
 int temp;
 temp = *a;
 *a = *b
 *b = temp;
printf ("After swap a = %d, b = %d", *a, *b)
}
```

& = ampersand → it passes address

09.12.21

Experimentally prove this,
int a, b
a = 10
b = a.
b = 10

Date 07/12/21
Page 85

* storage, classes and specifiers

scope of identifiers
→ local scope
→ Global scope
→ File scope

* Global scope :-

→ Identifiers which are declared outside any function have global scope.
→ Such identifiers are accessed by any part of the program.

```
#include <stdio.h>
#include <conio.h>
int a;
void main()
{
a=10;
printf("%d", a);
fun(a); getch();
}
fun(a=20);
printf("%d", a);
}
```

output
a=10
a=20

* Local scope :-

Identifier which are declared inside any function have local scope.
→ Such identifiers are accessible to the block statement block in which

file - initialise its value

Date
Page 86

they are declared as well as all inner statement blocks.

coding :-

```
void main()
{
int num = 10;
clrscr();
printf("Numbers are");
printf("\n num=%d, num);
}
int num=20;
printf("num=%d", num);
}
printf("num=%d", num);
getch();
}
```

output
Numbers are → num=10
num=20
num=20 10

* File scope :-

Variable declared outside main function with static keyword.

```
#include <stdio.h>
int global = 5;
void display()
{
printf("%d\n", global);
}
int main()
{
printf("Before change within main:");
```

```
display();   // call the function
printf("After change within main:");
global=10;
display();
```

Output

Before change within main : 5
After change within main : 10

* storage class specifier

→ Types of storage class

- Auto
- Register
- static
- External

→ It stores information about memory

→ life time

Memory location → help to fetch the data

scope → reduce the scarcity of variable

default value → store different value for different storage

10/12/21

* Auto storage class :-

→ The keyword auto is used to refer automatic variable.

→ syntax

```
auto int i;
```

→ Automatic variables are stored in memory.

→ initial value of automatic variable is a garbage value.

→ it is local to function.

→ when the variables are created and when the functions are called, and destroyed when that function terminated.

* Register storage class :-

→ It is stored inside CPU register in order to provide fast access.

→ syntax

```
register int a;
```

→ default initial value of register variable is garbage value.

*→ The difference between an auto variable and register variable is that you can not take the address of register value because hardware registers mostly do not have memory address on the other hand you can take the address of any variable.

*→ If the available registers are not enough due to any reason, the variables are treated as automatic variable.

→ Register storage class is restricted to integer variables only. because CPU registers are 16 bit registers and each integer occupies 2 bits.

* External storage class:

→ External variables are also referred as global variables.
→ External variables are created outside all function.
→ syntax
Extern int a;
→ it is also stored in memory.
→ default value is zero.
→ External variables are not restricted to any function locally.
→ life of external variable remains present throughout the program and it is destroyed only when the program terminates.

make code for adding in External.

```
# include <stdio.h>
# include <conio.h>
extern int a, b;
void main()
{
  a=10;
  b=20;
  int add;
  clrscr();
  avg=(a+b)/2;
  printf("Average=%d", add);
  getch();
}
```


* static storage class

→ There are two types of static variable.
(i) static local variable (Internal static variable)
(ii) static global variable (external static variable)

static local variables: are declared inside the function using a keyword static.
Syntax:
static int c;

The value of static local variable couldnot be destroyed even the function terminates.

static local variable can be accessed within its owns function

Storage class	keyword	Storage	default Initial value	Scope	Life
Automatic	Auto	Memory	Garbage value	Local to Block	Temporary
Register	Register	CPU register	Garbage value	Local to Block	Temporary
External	Extern	Memory	zero	Global to all files	Persistent
Internal static storage class	static	Memory	zero	local to all block	Persistent
External static storage class	static	Memory	zero	Global to one file	Persistent

* MACRO SUBSTITUTION :-

→ If an identifier is used to represent the statement or expression we call it macro.

→ A macro defines a meaning for an identifier

There are two types of macros :-

(i) object like macros
(ii) function like macros

(i) object like macros :- c provide # define pre processor directive to support macro facility

Syntax :-

define macro-name substitution-text;

Ex :-

define Π = 3.14

(ii) function like macros :-

the macros which have arguments so that they may look and act like function call are called function like macros.

Syntax :-

define macro-name (list-of-argument) substitution

* LINKED LIST

(A) SINGLY LINKED LIST :-

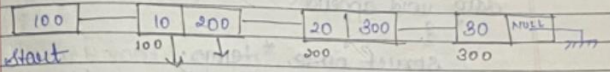
A link list is a linear data structure.

→ It is a collection of data elements called node.

→ Every node has two part :-

(i) information
(ii) pointer / address part

which link next element in the list.



Advantages over array :-

(i) Dynamic size.
(ii) ease of insertion and deletion.

Drawback :-

(i) Random access is not allowed.
(ii) extra memory space for a pointer is required with each elements of the list.

* CREATION OF NODE

struct node → (above int main)
{

```
int data;
struct node * link;
};
struct node * root = NULL;
```

append → to add anything in list

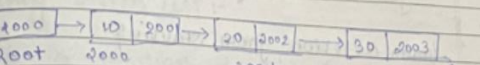
→ :- to access structures data member

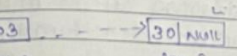
allocate memory

for insertion and adding:-

```
void append()
{
struct node * temp;
temp = (struct node *) malloc (sizeof(struct node));
printf ("Enter node data");
scanf ("%d", &temp->data);
temp -> link = NULL;
if (root == NULL)
{
root = temp;
}
else
{
struct node * p;
p = root;
while (p -> link != NULL)
{
p = p -> link;
}
p -> link = temp;
}
}
```


Dry Run

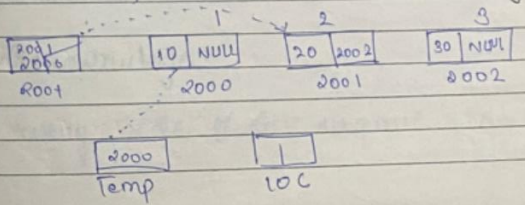




for deletion :- from starting

```
void delete()
{
struct root * temp;
int loc;
printf ("Enter location to delete");
scanf ("%d", &loc);
if (loc > length(list))
{
printf ("Invalid location");
}
else if (loc == 1)
temp = root;
root = temp -> link;
temp -> link = NULL;
free (temp);
}
```

Dry Run



you deletion fo from between

{

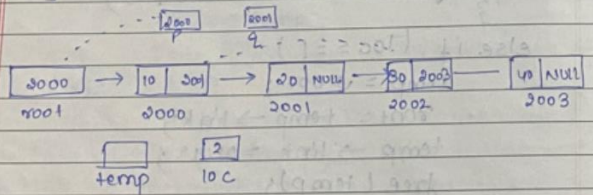
from deletion from starting

}

else

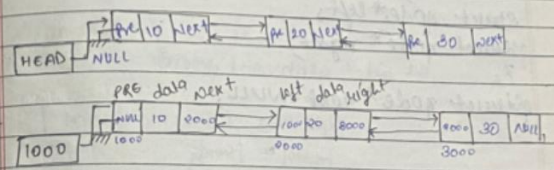
{

```
struct node *p = root, *q;
int i = 1;
while ( i < loc-1)
p = p -> link;
  i ++;
}
q = p -> link;
p -> link = q -> link;
q -> link = NULL
free (q);
}
}
```



* Doubly LINKED LIST :-

→ It is a variation of linked list in which navigation is possible, in both ways : i.e., forward and backward.



NEXT :- Each link of a linked list contains a link to the next link called "next".

PREVIOUS :- (PRE) :- Each link of a ~~list~~ link list contains a link to the previous link called "PRE".

Advantage :-

① It can be traverse in both forward and backward direction.

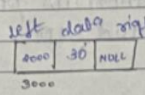
Disadvantage :-

① Every node of DLL require extra space

for an previous pointer.

Creating Node

```
struct node
{
int data;
struct node* left;
struct node* right;
};
struct node* root = NULL;
```



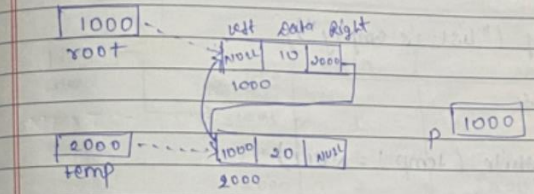

```
void append()
{
struct node* temp;
temp = (struct node*) malloc(sizeof(struct node));
printf("Enter node data");
scanf("%d", &temp->data);
temp->left = NULL;
temp->right = NULL;
if (root == NULL)
{
root = temp;
}
else
{
struct node* p;
p = root
while (p->right != NULL)
```



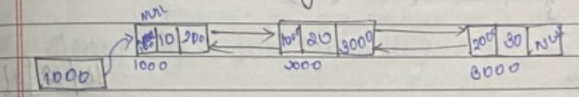
```
{
p = p->right;
}
p->right = temp;
temp->left = p;
}
}
```

Dry Run :-

Enter the data : 10, 20



* LENGTH OF Doubly Linked list




```
int length()
{
struct node* temp = root;
int count = 0
while (temp != NULL)
{
count ++;
temp = temp->right;
```

NULL
temp

3
count

```
}
return count;
}
```

* code for display all the elements in DLL

```
int display()
{
struct node * temp = root;
if (temp == NULL)
{
p.f ("List is empty");
}
else
{
while (temp != NULL)
{
p.f ("%d →", temp → data);
temp = temp → right;
}
}
}
```

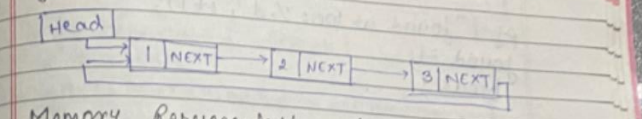
16/12/21

* CIRCULAR LINKED LIST

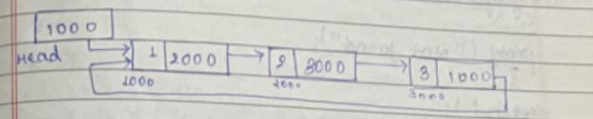
→ In this linked list the last node of the list contains a pointer to the first node of the list.

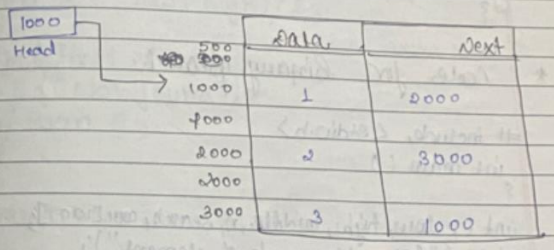
There is no null value present in the next part of any of the nodes.

Syntax:-



Memory Representation of a Circular linked list:





* Searching and Sorting

17/12/21

* linear search

coding :-

```
void main()
{
int ele, found = 0;
for (i=0; i<6; i++)
{
if (arr[i] == ele)
```

```
}
pf ("found at loc: %d", i+1);
found = 1;
break;
}
}
if (found == 0)
{
printf ("Not found");
}
getch ();
}
```

* Code for Binary search

```
#include <stdio.h>
int main ()
{
int i, low, high, middle, n, search, arr[100];
printf ("Enter the no.'s of element");
scanf ("%d", &n);
for (i=0; i<n; i++)
{
scanf ("%d", &arr[i]);
}}
p.f ("Enter value to find.");
sf ("%d", &search);
low = 0;
high = n-1;
middle = (low + high)/2;
```



```
while (low <= high)
{
if (arr[middle] < search)
low = middle + 1;
else if (arr[middle] == search)
{
p.f ("%d found at location %d", search, middle+1);
break;
}
else
last = middle - 1;
middle = (low + high)/2;
}
if (first > last)
p.f ("Not found");
return 0;
}
```

* SORTING

→ arrangement of data either in ascending order or descending order.

Bubble Sorting

		0	1	2	3	4
i=0		50	40	30	20	10
Pass=I	i=1	40	30	30	20	10
Pass II	i=2	40	30	50	20	10
	i=3	40	30	20	50	10
	i=4	40	30	20	10	50

Date 03/01/22
Page 103

Main code:-

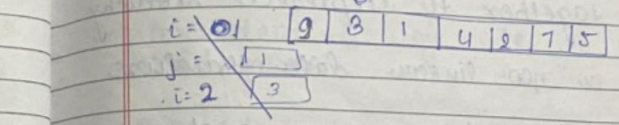
```
int main()
{
int n, temp, i, j, arr[100];
printf("How many no's you will enter");
scanf("%d", &n);
printf("Enter %d number:", n);
for(i=0; i<=n; j++)
scanf("%d", &arr[i]);
for(i=0, i<n-1; i++)
{
for(j=0; j<n-1, j++)
{
if[arr[j] > arr[j+1]]
{
temp = arr[j]
arr[j] = arr[j+1];
arr[j+1] = temp;
}}
}}
```

* selection sorting

```
int main()
{
int i, j, min, temp;
for(i=0; i<n-1; i++)
{
min = i;
for(j = i+1; j<n; j++)
```

Date
Page 104

```
{
if(a[j] < a[min])
{
min = j;
}
}
temp = a[i];
a[i] = a[min];
a[min] = temp;
}
```



* Insertion sort

```
for(i=1; i<7; i++)
{
temp = a[i];
j = i-1;
while(j>=0 && a[j]<temp)
{
a[j+1] = a[j];
j = j-1
}
a[j+1] = temp;
}
```

* TREE

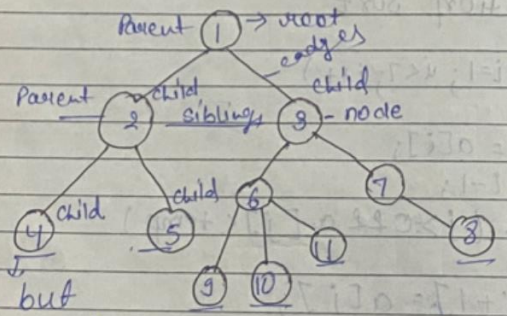
→ also known as "Recursive data structure"

→ A tree is also one of the data structure that represent ~~hire~~ ~~hierarchical~~ ~~data~~ here hierarchal data.

→ It is a collection of ~~of~~ objects or entities known as nodes that are link together to represent hierarchy.

→ It is a non-linear data structure

→ top-most node is known as root node.



node but
leaf node

no of nodes = 11 n
no. of edges = 10 n-1
no. of internal node = 5

→ Each node contains some data and the link or reference of another nodes that can be called "children".

→ arrows shows the link between the two nodes.

*→ Root is the top most node in the tree hierarchy

→ Root node does not have any parent

* Child node

→ discendent of any node is known as child node

* Parent node

→ A node containing any sub-node, then that node is called "parent node."

* Siblings

→ node that have same parent are called siblings.

* Internal nodes :-

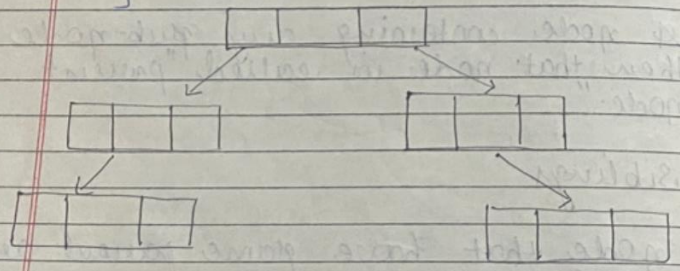
→ A node that has atleast one child node is known as internal node.

* Height of node :-
Longest path from the node to the leaf node

04/01/22

* Implementation of tree :-
Tree data structure can be creating nodes dynamically by using pointers.

```
struct node
{
    int data;
    struct node * left;
    struct node * right;
};
```



Application of Tree :-

(1) storing naturally heirarchal data e.g. file system

(2) organize data

(3) Trie :- special type of tree used to store dictionary.

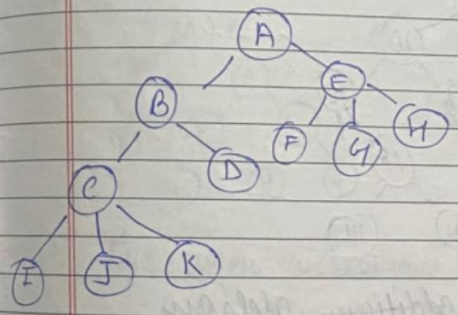
Heap :- It is implemented using array use to implement priority queue

B-tree & B+ tree :-
used to implement indexing in data base

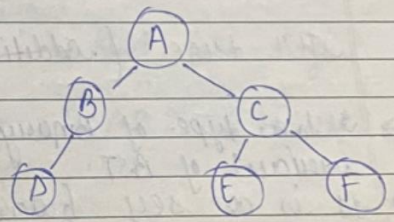
Routing table :-
use to store the data in the routing tables in the router.

* Types of Tree data structure

(1) General Tree



BINARY TREE



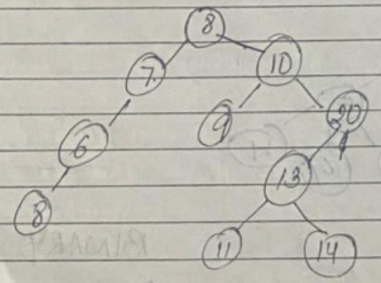
Each node is a tree which have almost two child node. (0,1,2)

* BST (Binary search Tree :-

BST is a ~~known~~ now linear dat structure in which one node is connected to n number of node.

Every node in left sub tree must contain a value less than the value of root node and the value of each node in the right tree must be bigger then the value of the root node.

Q. 8, 10, 20, 13, 14, 9, 7, 6, 5, 11



AVL tree (~~Addition~~ Adelson

→ It is a type of Binary tree and it is a variant of B.S.T.

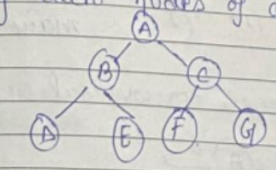
→ It is a self balancing binary research

tree.

Balancing factor (value) is either 0, -1, 1

* Tree transversal. 06/01/22

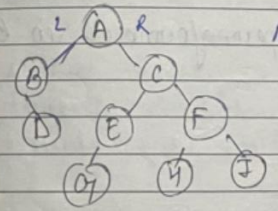
Visiting each nodes of a tree.



Different types of transversal

(i) In order transversal
(ii) ~~Tree~~ Pre order transversal
(iii) Post order transversal
(iv) Level by level

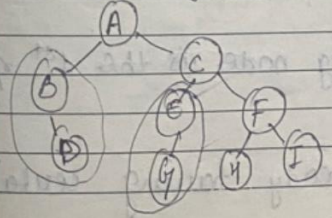
In order Transversal



D B A G E C H F I

Left → Root → Right

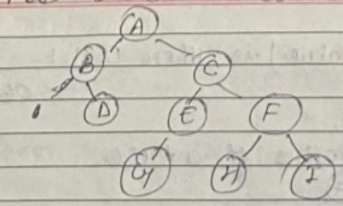
Pre-order Transversal



Root - left - Right

A B D C E G F H I

(iii) Post-Order traversal

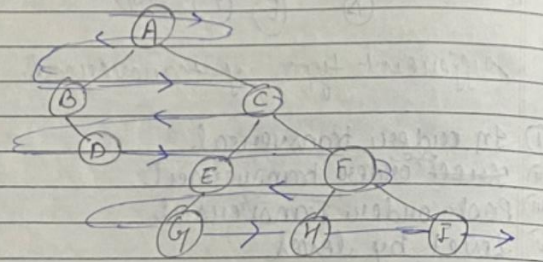


Left - Right - Root

D B G E H I F C A

main root = A

* Level by Level traversal 18/01/22



A B C D E F G H I

* Various operation performed in BST

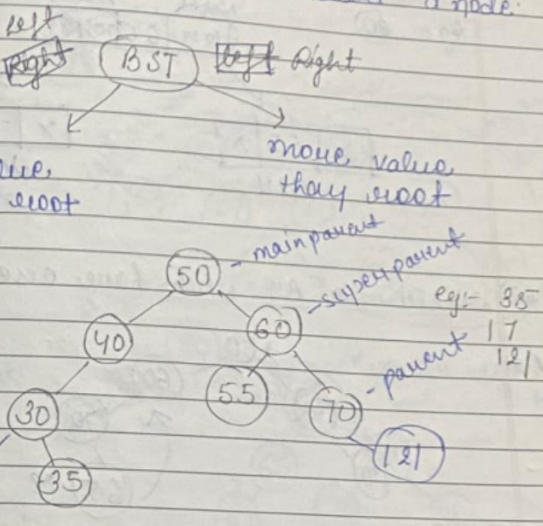
→ (i) Insertion of node
→ (ii) Deletion of node
→ (iii) Searching of node

* Insertion of node in BST

→ when the inserting node is the 1st node of the BST.

Imp → when BST is already having certain

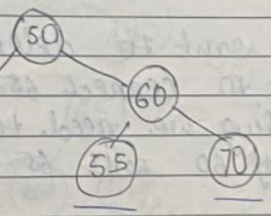
nodes and we wish to insert a node.



* Deletion of node in BST

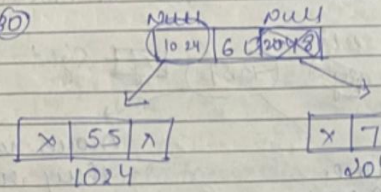
0 → when there is no child
(i) → Having single child
Imp → Having two children

(i)

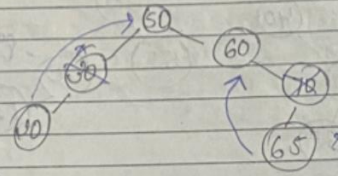


*(i) Deletion in BST which have no child

Eg:- 80

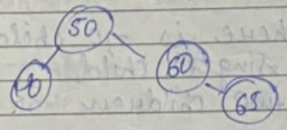


*(ii) Deletion in BST which have one child



Eg:- 30

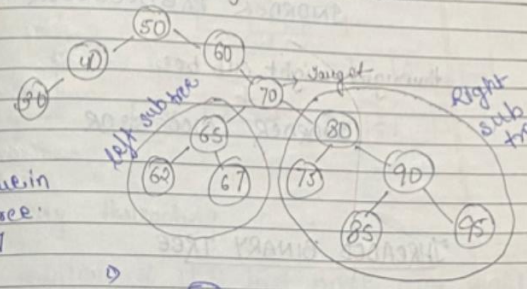
(i) 70 After deleting :-



If we want to delete 70 then we want to connect 65 to 60. But before connecting we need to check which side of 60 will 65 be connected.

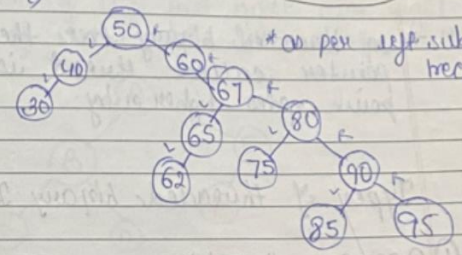
11-01-22

(iii) Deleting a node having two children



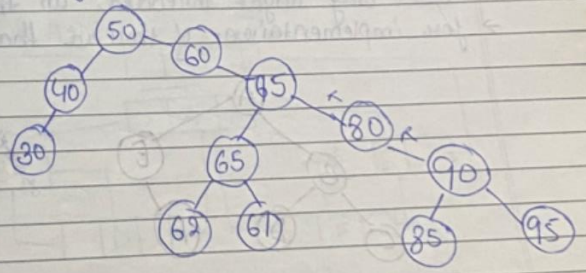
→ Highest value in left sub tree.
Eg:- 67

Right sub tree → least value in the right sub tree
Eg:- 75



*as per left sub tree :-

as per right sub tree :-



through left sub tree
↓
INORDER PREDECESSOR

through right sub tree
↓
INORDER SUCCESSOR

12-01-21

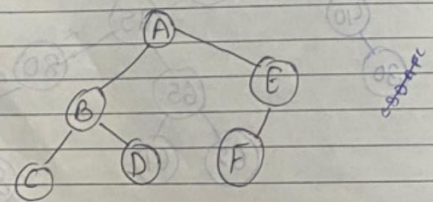
THREADED BINARY TREE

→ In threaded binary tree the special pointer called thread is used to point nodes higher entry.

Types of threaded binary Tree

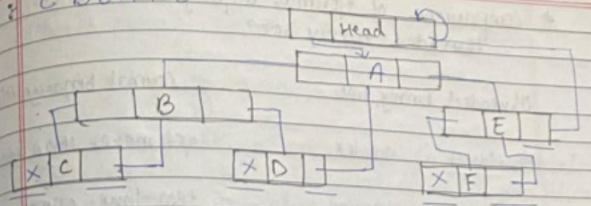
① One way threading

→ use only right threaded in this case
→ for implementation of thread that in order



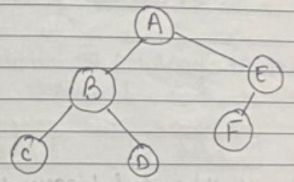
① Inorder (left root Right)

: C B D A F E



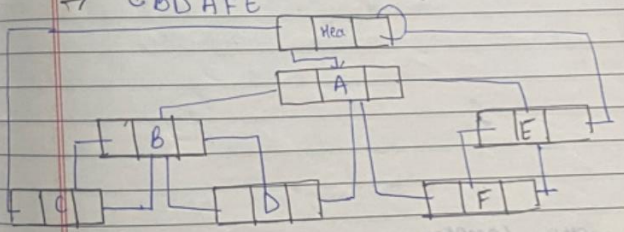
② Two way threading

left pointer of the first node and right pointer of that node will contain null values



Inorder (left Root Right)

→ C B D A F E



input output → scanf(" ")] predefined function
output input → printf(" ")

stdio → standard. library
↳ standard. input output
conio → console. input output

* main() function tells the computer where the program. start.
→ Every program have exactly one main function
→ if there will be more than one function compiler will not understand which one mark the begining of the program.
→ /* */ → comment lines //→ single line comment
↳ non-executable statements, ignored by compiler
→ enhances the readability and understanding
* predifined function it is a function that has already been written and compiled and linked together with our program at the time of linking
getch(); → to hold the screen
%d → integer data type
for int ↳ format string
for float → %f
local → it will create as many as memory as we create.
not all identifier are variable

Q: distinguish between
→ main() and void main(void)
→ int main() and void main()

Structure	Union
structure st1	union st1
{	{
int a;	int a;
char b;	char b;
float c;	float c;
}	}
make own memory	share memory to the largest one.
4 byte | 1 byte | 4 byte	4 byte
a b c	a b c

printf("\n Data swapped successfully...");
printf("Data in list after swapping %d node with %d:\n")
display list();
}
else
{
printf("Invalid position, please enter position less <0 or >1 in list:\n")
}
return 0;
}

TOP- Down
Programming
Stack
delete string
Pointer types

function
Array
Stack
Queue
Operator and its type
Tree
Binary tree with its type
Circular queue
Recursion
Structure

%.d → integer data type
for int ↳ format string
for float → %.f
local → it will create as many memory as we create
not all identifier are variable

```
# include <stdlib.h>
# include <stdio.h>
struct node
{
  int data;
  struct node *next;
};

int main()
{
  int n, pos1, pos2;
  printf("Enter number of node to create:");
  scanf("%d", &n);
  create list(n);
  printf("\nEnter first node position to swap:");
  scanf("%.d", &pos1);
  printf("\nEnter second node position to swap:");
  scanf("%.d", &pos2);
  if (swap(head, pos1, pos2) == 1)
  {
    printf("\n Data swapped successfully.\n");
    printf("Data in list after swapping %d node with %d: \n")
    display list();
  }
  else
  {
    printf(" Invalid position, please enter position <0 or >1 in list.\n")
  }
  return 0;
}
```

algorithm for

1 - (iii)
2 - (ii)
3 - (iii)
4 - (i) *
5 - (ii)
6 - (iii)
7 - (i)
8 - (iii)
9 - (ii)
10 - (i)

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

```
main ( )
{ add (int a, int b)  → actual param
  { return ...
void add (int a, int b) → formal parameter
  p- ( ... )

# include
# include
void add (int a, int b);  → actual para
void main ( )
{
  int a = 10, b = 20
  add (a, b);        → actual parameter
                     → the value is pass during calling of funct
  getch ( );
}
void add (int a, int b) → formal parameter
{                          → variable are limited
  int c;
  c = a + b;
  printf ("sum = %d", c);
}
```

* → points to data/value
* → of pointer show the address

Call by value	Call by address
→ ~~value doesn't changes~~	→ ~~value~~ address changes
→ we pass the values	→ ~~add~~ pass address of function
→ function pass get copied in	→ address doesn't get copied

vanitta - 15Q   Q → 15Q

Q formal parameter and actual parameter
Q unformatted input and formatted input-output

(iii) Q: data structure and its type

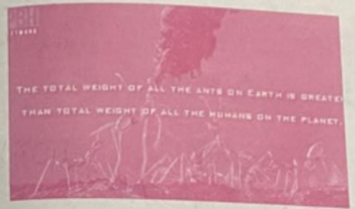
Ex:- → array, ~~tree~~ stack, queue, linked list, tree
→ store data in organized or efficient manner/way.
→ multiple data in organized form
two types:-
linear data str. → array, stack
→

non-linear → tree, graph

(ii) unformatted input-
(i) used for single character
Ex:- void main()
{
char name [5];
printf ("Enter the name");
~~scanf~~ getchar(name); and getch
output · unformatted → putchar

formated input → scanf
formated output -

If the human brain were a computer, it could perform 38 thousand-trillion operations per second. The world's most powerful supercomputer, BlueGene, can manage only



The total weight of all the ants on Earth is greater than total weight of all the humans on the planet.

Amazing Facts

Starfish can re-grow their arms. In fact, a single arm can regenerate a whole body.

The average hummingbird's heart rate is more than 1,200 beats per minute.