Time Complexity

Prime → 2 factors

others

→ for i → 2 to n-1 if divides by n it's not prime no.

→ 11 - 2 = 9ms

n = 101 ms.

→ $n = 10^6 + 3 = 10^6$ ms

$\approx 10^6$ (sec) → 16.66 mins

$10^{10} + 19 = 10^{10}$ ms

⇒ 10^{10} sec ⇒ 115 days.

Sir

for i → 2 $\sqrt{n}$ if i divide by n it is not prime

$\sqrt{11}$ → 3 - 1 = 2 ms

→ $\sqrt{101}$ → 9 ms

$\sqrt{10^6} + 3 - 1 = 10^3$ ms
= 1 sec

$\sqrt{10^{10}} + 3 - 1 = 10^3$ ms = 1 sec

insertion → tree
retraiving → hashing

24/11/2016

DATA STRUCTURES

Defination:-

1 "The logical and mathematical representation of data in the computer memory is called as data-structure".

2 "A data structure is a method of storing data in a computer so that it can be used efficiently."

⇒ The data structures mainly deals with the study of how the data is organised in the memory.

⇒ How efficiently the data can be stored in the memory.

⇒ How efficiently the data can be retrived and manupulated

→ Data Structures it is tells you the representation of logical relationships between elements of data

→ In other words a data structure is a way of organising data items by considering its relationship to each other.

→ Data structures affects the design of both structural and functional aspects of a program.

$\boxed{\text{Algorithm + data Structure = Program}}$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

→ The representation of particuler data-structure is the memory of a computer is called as storage structure i.e a data structure should be representated in such a way i.e utilized maximum efficiency.

→ The data structures can be maintained in both main memory and ~~(secon)~~ auxillary memory of the computer.

→ A storage structure representation in auxillary memory is ~~offt~~ often known as "file structure".

→ finally that the data structures and the operations on organized data items can combinly solving the problem using computer.

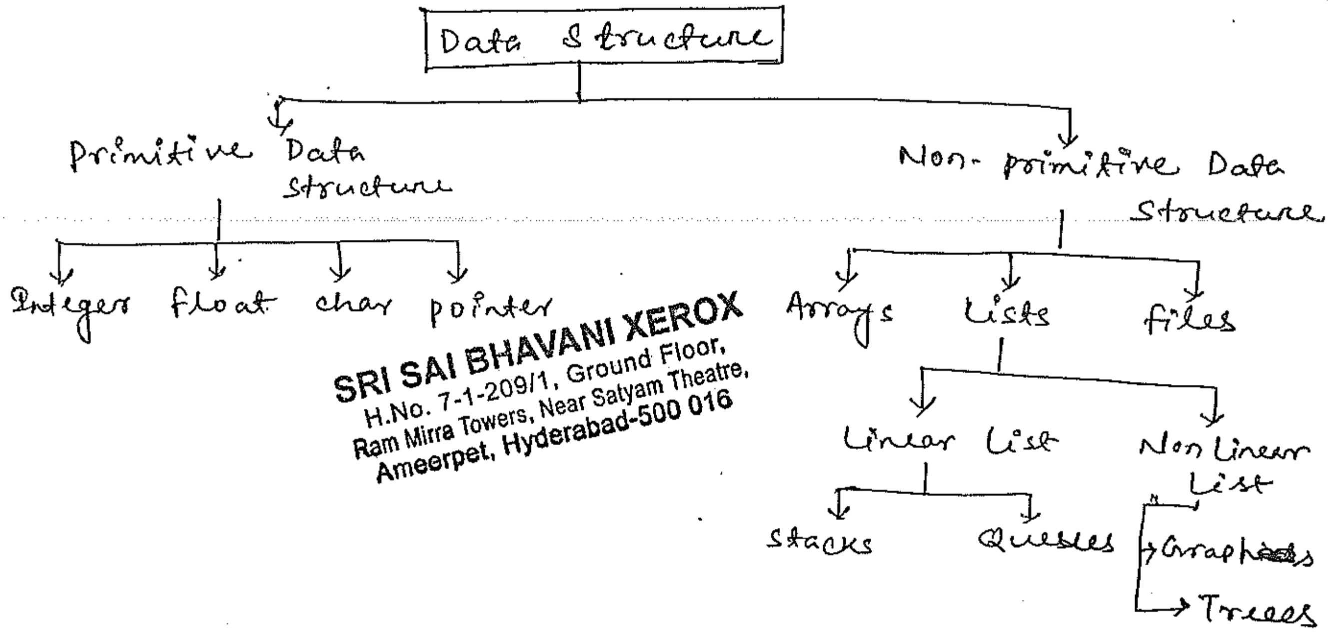
$$\boxed{\text{Data Structure} = \text{Organized Data} + \text{Operations}}$$

→ The data structures are classified in two ways

1. Primitive Data structures
2. Non primitive data structures.

→ The primitive data structures are the data structures that can be manupulated directly by the machine statements.

→ Non primitive data structures cannot be manupulated directly by the machine statements. These non-primitives are of two types
a. Linear
b. Non-Linear.



→ In many areas data structures are mostly implemented like: compiler Design, Operating Systems, Data Management -systems, statistical Analysis package, Numerical -Analysis, Graphics, Artificial Intelligence, simulation etc.

→ The major Data structures using the following areas RDBMS, Network Data Model and Hierarchial Data Model

→ RDBMS ⟶ Array (i.e Array of structures)
→ Network Data Model → Graph.
→ Hierarchial Data Model → Trees.

25/11/2016

ARRAYS

As we know that primitive data structures like integer, float etc, they can store only small amount of data. we need to declare number of variables. for suppose, if i want to store marks in three subjects i, need to declare 3 variables. for n subjects, n memory locations required. Defining the memory locations is possible but it is not recomendable, since the input and output statements and logical statements are increasing. To overcome this problem we use non-primitive data structures.

Defination of Array

An array is an single subscripted variable which can hold. n numbers of values in the single variable

Example for static memory allocation is Arrays

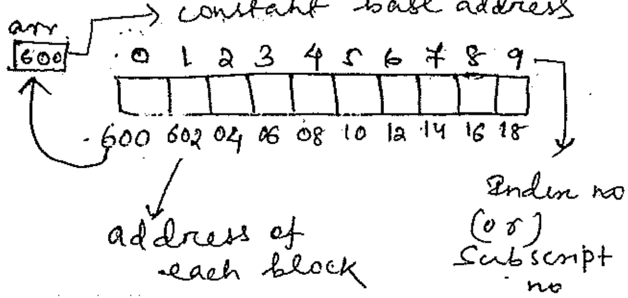
→ when we defining static memory allocations, the size of the array has to define. at the time of compilation.

→ without defining the size it is not possible to declare static memory. ~~alloco~~

Syntax:-

datatype var_name [size]; (size → mandatory)

Example :- int arr[10];



→ once we define the size of an array, you cannot increase or decrease the size.

→ An Array is an homogeneous type. (same type)

→ An array elements will be identified logically by the subscript numbers and phisically identified by the addresses

→ The array subscript allways starts with 'zero', for suppose 'n' is a size, then there will be (n-1) subscript numbers are existed.

Q) Write a program accept an array of elements and display that.

i = index No.
a[i] → Elements of the array
a → constant Base address

```
main()
{
  int a[100];
  int n, i;
  clrscr();
  printf(" Enter the size of n: ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
   {
    printf(" Enter the element [%d]: ", i);
    scanf(" %d", &a[i]);
   }
  for(i=0; i<
   {
    pr              , a[i]);
   }
} getch();
```


Q) Write a program accept an element, delete that element from an array.

```
main()
{
  int a[150];
  int n, i, de, status = 0, j;
  clrscr();
  printf(" Enter the size of n:");
  scanf("%d ", &n);
  for(i=0; i<n; i++)
   {
    printf("Enter the element at a[%d] : ", i);
    scanf(" %d", &a[i]);
   }
```

```
printf ("\n Enter the deleting element :");
scanf ("%d", &de);
printf("\n Before deletion the elements are ...\n");
for(i=0; i<n; i++)
printf ("%3d", a[i]);

for(i=0; i<n; i++)
 {   if (a[i]==de)
     {
      status = 1;
      for (j=i; j<n; j++)
          a[j] = a[j+1];
          i--; // for duplicates
      }   n--;
 }
if (status==0)
 printf("\n Element not found");
else
{ printf("\n Elements after deletion..\n");
  for (i=0; i<n; i++)
     printf("%3d", a[i]);
 }
 getch();
}
```

26/11/2016

Disadvantage :-

→ As an array is an homogeneous it can hold, only same type of elements.

→ Many applications in real time requires heterogeneous type of data.

→ As an array is an static memory allocation the size has to define at the time of compilation. once we define the size it is never possible to increase or decrease the size.

→ This all the problems can be overcome by linked list Concept.

STACKS

→ A stack is an special data structure where elements are inserted from one end and elements are deleted from the same end.

→ The position from where the elements are inserted and from where the elements are deleted is called as Top of the Stack

→ Stack is also called as last in fast out data structure.

→ stack application can be implemented in two ways

1. Arrays
2. Linked List.

→ The general operations which takes place on stack

(i) Push → Inserting an element on the top of the stack.

(ii) Pop → Deleting an element from top of the stack.

(iii) Display → Display the contents of the stack.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

→ when we implementing by the linked list concept the no of operations can be changes.

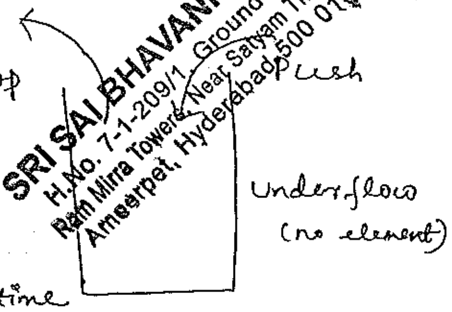
→ As we working with arrays concept we need to define the size of the array at the time of compilation.

MAX = 5 ↳ size

top = -1

→ initially stack is empty, it indecates underflow condition.

i.e there is no elements in the stack.



→ At the time of push operation everytime it checks the stack is overflow or not. if it is overflow we cannot implement push operation.

→ Every time of push operation the top will be incremented. and then push operation takes place where ever the top is pointing.

→ Insert 10

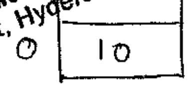
if (top == MAX-1) // overflow condition.

" Overflow "

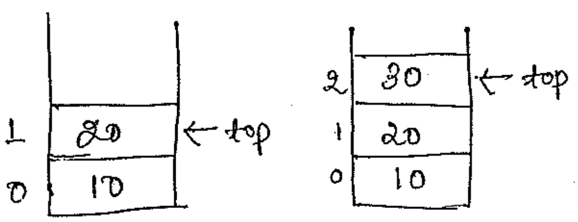
if this condition not ~~st~~ satisfied we can implement push operation by incrementing the top.

index ← top ++; // -1++ = 0

array ← stack [top] = ele ;



→ Insert an elements 20 & 30



⇒ if you implement pop operations, then it check the top condition

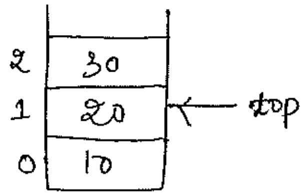
if (top==-1)

" Underflow"

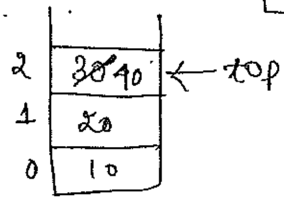
i.e there is no elements in the stack. (empty stack)

⇒ when ever the top operation takes place, where ever the top is pointing that element logically deleted in arrays by decrementing the top position.

⇒ It is logically deleted, physically existed.



⇒ Insert 40 in the stack



28/11/2016

```
#define MAX 4
int stack[MAX], top=-1, ele;
Void push();
Void pop();
Void print();
main().
{
 int ch;
 Clrscr();
 do
 {
  printf ("\n 1. push ");
  printf ("\n 2. pop");
  printf ("\n 3. print");
  printf ("\n 4. Exit");
  printf ("\n Enter the choice ");
  scanf ("%d", &ch);
  Switch (ch)
  {
   Case 1 : push(); break;
   Case 2 : pop(); break;
```

```
case 3: print(); break;
case 4: exit(0);
}
} while (ch != 4);
getch();
}
```



```
void push()
{
if (top == MAX-1)
printf("\n overflow");
else
{ printf("Enter the element :");
  scanf("%d", &ele);
  top++;
  stack[top] = ele;
}
}
```



```
void pop()
{
  if (top == -1)
  printf("Underflow \n");
  else
  { ele = stack[top];
    top--;
    printf("\n The deleted element is : %d", ele);
  }
}
void print()
{
  if (top == -1)
  printf("stack empty \n");
  else
  { for (i = top ; i >= 0 ; i--)
     printf("%d", stack[i]);
  }
}
```

Applications of stacks :-

(→ push
) → pop

⇒ Based on the stack decipline ie Last in fast out, number of applications like, balancing of symbols, conversion of expressions (Infix to postfix etc),

⇒ Evaluations of the expression.

⇒ Implementing function calls (Including Recursion)

⇒ Undo sequences in a text editor (ctrl +z)

⇒ page-visited History in a web browser (Back botton)

⇒ finding of spans (stock market)

⇒ Matching Tags in HTML and XML etc.

Conversions and Evaluation of Expressions :-

* what is an expression?

An expression is a combination of sequence of operators and operands that reduces to a single value after evaluation is called as an expression.

This expressions are of 3 types

1. Infix expression
2. prefix expression
3. postfix expression.

1. Infix expression :-

In a expression if an operator is placed inbetween the two operands, the expression is called infix expression.

Example :- (A + (B - C))

a + b * c

2. prefix expression :-

In an expression if an operators placed before the operands such type of expression called as prefix expression.

Example :- + a - bc

+ a * bc

3. postfix expression :-

In an expression operands 1st and then operators such type of expression is called as postfix expression.

Example :- abc − +

abc * *

Note :- Always the compilers try to convert the expression into the postfix expression.

⇒ Infix to prefix conversion :-

(a+b) * (a−c)

The below table shows arithmatic operators along with priority values and their associativity.

Description	operator	Priority	Associativity
Exponentiation	\$ or ^	6	Right to Left
Multiplication	*	4	Left to Right
Division	/	4	L to R
Mod	%	4	L to R
Addition	+	2	L to R
Substraction	−	2	L to R

$(a+b) * (a-c)$

$\underbrace{+ab}_{x} * \underbrace{-ac}_{y}$ [x & y are operands now]

$* +ab-ac$

When

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Note :-

1. Whenever an expression is converting, the expression after conversion treated as an operand.
2. At the time of conversion do not change the order of the operands.

29/11/2016

$(A+(B-C))$

$(A+\underline{-BC})$

$\underline{+A-BC}$

$(A+(B-C)*D)$

$(A+\underline{-BC}*D)$

$(A+\underline{*-BCD})$

$\underline{+A*-BCD}$

Infix to postfix :-

$(a+b)*(a-c)$

$\underline{ab+}*\underline{ac-}$

$\underline{ab+ac-*}$

$(A+(B-C))$

$(A+\underline{BC-})$

$\underline{ABC-+}$

$(A+(B-C)*D)$

$(A+\underline{BC-}*D)$

$A+\underline{BC-D*}$

$\underline{ABC-D*+}$

$((A+(B-C)*D)\wedge E+F)$

$((A+\underline{BC-}*D)\wedge E+F)$

$((A+\underline{BC-D*})\wedge E+F)$

$(\underline{ABC-D*+E\wedge}+F)$

$\underline{ABC-D*+E\wedge F+}$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Algorithm 1 :-

Infix to post-fix Conversion :-

scan from L to R Repeat step 1-4

Token	operation
operand	Add to expression
(	Push to stack
operator	pop operation , if P(popped) >= P (scanned) add to expression . Push scanned operator to stack
)	pop till (Add popped tocken to expression , Delete (

Pop stak elements if any add to expression

A + (B * C - (D / E $ F) * G) * H

Token	Stack	Expression
#	#	
A	#	A
+	# +	A
(	# + (	A
B	# + (	AB
*	# + (*	AB
C	# + (*	ABC
-	# + (-	ABC*
(	# + (- (	ABC*
D	# + (- (	ABC*D
/	# + (- (/	ABC*D
E	# + (- (/	ABC*DE

Popped → top of stack
Scanned → new

#

scanned

popped # >= +
0 >= 2 X

(
+
#

(>= *
0 >= 4 X

* >= -
4 >= 2 ✓

(>= /
0 >= 4 X

$	# + (− C / $	ABC * DE
F	# + (− C / $	ABC * DEF
)	# + (−	ABC * DEF $ /
*	# + (− *	ABC * DEF $ /
G	# + (− *	ABC * DEF $ / G
)	# +	ABC * DEF $ / G * −
*	# + *	ABC * DEF $ / G * −
H	# + *	ABC * DEF $ / G * − H
		$\boxed{\text{ABC * DEF \$ / G * − H * +}}$

A + (B * C − (D / E $ F) * G) * H

A + (B * C − (D / EF$) * G) * H

A + (B * C − DEF$/ * G) * H

A + (BC* − DEF$/ * G) * H

A + (BC* − DEF$/G*) * H

A + BC*DEF$/G*− * H

A + BC*DEF$/G*−H*

$\boxed{\text{ABC * DEF \$ / G * − H * +}}$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

A $ B * C - D + E / F / (G + H)

Token	Stack	Expression
#	#	
A	#	A
$	# $	A
B	# $	AB
*	# *	AB$
C	# *	AB$C
-	# -	AB$C*
D	# -	AB$C*D
+	# +	AB$C*D-
E	# +	AB$C*D-E
/	# +/	AB$C*D-E
F	# +/	AB$C*D-EF
/	# +/	AB$C*D-EF/
(	# +/(	AB$C*D-EF/
G	# +/(	AB$C*D-EF/G
+	# +/(+	AB$C*D-EF/G
H	# +/(+	AB$C*D-EF/GH
)	# +/	AB$C*D-EF/GH+
		$\boxed{AB\$C*D-EF/GH+/+}$

$
\#

\# >= $

0 >= 6

$ >= *

* >= -

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

30/11/2016

// program on Infix to postfix (suffix) conversion.

```
#include <stdio.h>
#include <ctype.h>
#define MAX 20
char infix[MAX], post[MAX], s[MAX], ele, ch, x, t;
int i=0, top=-1, j=0;
void push (char ch)
{
 top++;
 s[top] = ch;
}
char pop()
{
  ele = s[top];
  top--;                    return (s[top--]);
  return ele;
}
int priority (char ch)
{
  if (ch=='^')
      return 4;
  else if (ch=='*'|| ch='/'||ch=='%')
      return 3;
  else if (ch=='+'|| ch=='-')
      return 2;
  else
      return 0;
}
void check()
{
  while (priority(t) <= priority (s[top]))
     post[j++] = pop();
}
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
main()
{
 clrscr();
 printf("Enter the Infix Expression :");
 scanf("%s", infix);
 push('#');
 while (infix[i] != '\0')
 {
  t = infix[i];
  if (isalpha(t))
  post[j++] = t;
  else
  {
   if (t=='+' || t=='-' || t=='*' || t=='/' || t=='^'
                || t=='%' || t=='(' || t==')')
     switch(t)
     {
     case '(' : push(t); break;
     case '+' :
     case '-' :
     case '%' :
     case '/' :
     case '*' :
     case '^' :     check();
                    push(t);
                    break;
     case ')' : do
                {
                 x = pop();
                 post[j++] = x;
                }
                while (x != '(');
                j = j-1;
                break;
     } //switch
  } // else
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
    i = i+1;
  } // while
  while (s[top]! = '#')
    {
      post[j++] = pop();
    }
    post[j] = '\0';
  printf("The postfix Notation of given infix is %s", post);
  getch();
} // main
```

O/P→ Enter the Infix Expression : (a+b)*(a-c)
The postfix Notation of given infix is ab+ac-*.

1/12/2016

Evaluation of postfix Expression :→

→ As the compiler make a conversion into the postfix it is not necessery to give the priorities to the operators. Postfix has to evaluation based on the algorithm.

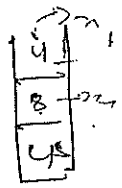
→ Scan from L to R Repeat step 1-2.

Token	Operation
Operand	Add to stack
operator	pop stack into n1 Pop stack into n2 perform n3 = n2 operator n1 (order) push n3

pop stack to obtain the result.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

4 2 \$ 3 * 3 − 8 4 / 1 1 + / +



Token	Stack
4	4
2	4, 2
\$	16
3	16, 3
*	48
3	48, 3
−	45
8	45, 8,
4	45, 8, 4
/	45, 2
1	45, 2, 1
1	45, 2, 1, 1
+	45, 2, 2
/	45, 1
+	46

$n_1 = 2 \quad n_2 = 4$

$n_3 = n_2 \text{ operator } n_1$

$= 4 \wedge 2 = 16$

$n_1 = 4, \quad n_2 = 8$

$n_3 = n_2 / n_1$

$= 8/4 = 2$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Infix → ((A + (B − C) * D) \$ E + F)

postfix → A B C − D * + E \$ F +

with the values

A = 6, B = 3, C = 2, D = 5, E = 1, F = 7

Token	stack
A	6
B	6, 3
C	6, 3, 2
−	6, 1
D	6, 1, 5
*	6, 5
+	11
E	11, 1
$	11
F	11, 7
+	18

Program on post fix evaluation.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<math.h>
#define MAX 50
char suf[MAX], ch;
float s[MAX], op1, op2, op, val, temp, res;
int i=0, j=0, top=-1;
float pop()
{
  return (s[top--]);
}
void push(float val)
{
  top++;
  s[top]=val;
}
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
float operate (float op1, float op2, char ch)
 {
   switch (ch)
    {
      case '+' : temp = op1 + op2;
                 break;
      case '-' : temp = op1 - op2;
                 break;
      case '*' : temp = op1 * op2;
                 break;
      case '/' : temp = op1 * op2;
                 break;
      case '^' : temp = pow ( op1 , (int) op2 );
                 break;
    }
    return temp;
 }
main ()
{
  clrscr();
  printf (" Enter the suffix expression :");
  scanf (" %s", suf)
  while ( suf[i] !
   {
     ch = suf[i];
     if ( isalpha(ch))
      {
        printf (" Enter the value for %c =", ch);
        scanf (" %f ", &val);
        push ( val);
      }
     else
     if ( ch == '+' || ch == '-' || ch == '*' || ch == '/' ||
                          ch == '%' || ch == '^').
      {
```

```
        op2 = pop();
        op1 = pop();
        res = operate (op1, op2, ch);
        push(res);
      }
   }
  i = i+1;
  } // while
 temp = pop();
 printf (" The simplified answer for %s = ", suf);
 printf(" %f ", temp);
 getch();
} // main
```


O/P →

Enter the suffix expression : ABC * − D * + E ∧ F +

Enter the value for A = 6

Enter the Value for B − 3

" " " " C = 2

" " " D = 5

" " " E = 1

" " " F = 7

The simplified answer for ABC − D * + E ∧ F + = 18.000000.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Algorithm - 3 :-

Infix to prefix conversion.

Scan from R to L repeat step 1 - 4

Token	operation
operand	Add to expression
)	Push to stack
operator	pop operator if P(popped) > P(scanned) add to expression. push scanned operator to stack.
(	pop till) . Add popped token to expression. Delete)

pop stack elements if any and add to expression.

1/12/2016

A + (B * C - (D / E $ F) * G) * H   R → L

Token	Stack	Expression	
#	#		
H	#	H	
*	# *	H	
)	# *)	H	
G	# *)	HG	> GH
*	# *) *	H G	GH
)	# *) *)	H G	GH
F	# *) *)	HGF	FGH
$	# *) *) $	HGF	FGH
E	# *) *) $	HGFE	EFGH
/	# *) *) /	HGFE$	$EFGH

> *

(> y

$ > /

D	#*)*)/	HGFE$D
(	#*)*	HGFE$D/
-	#*)-	HGFE$D/*
C	#*)-	HGFE$D/*C
*	#*)-*	HGFE$D/*C
B	#*)-*	HGFE$D/*CB
(	#*	HGFE$D/*CB*-
+	#+	HGFE$D/*CB*-*
A	#+	HGFE$D/*CB*-*A
		#HGFE$D/*CB*-*A+
		+A*-*BC*/D$EFGH

*Y-

A+(B*C-(D/E$F)*G)*H

A+(B*C-(D/$EF)*G)*H

A+(B*C-/D$EF*G)*H

A+(*BC-/D$EF*G)*H

A+(*BC-*/D$EFG)*H

A+-*BC*/D$EFG*H

A+*-*BC*/D$EFGH

+A*-*BC*/D$EFGH

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

→ At the time of implementing the program after accepting the input of infix expression 1st reverse the string and implement the program at time of displaying the output. once again reverse and display.

Algorithm - 4 :-

Evaluate prefix

scan from R to L repeat step 1 - 2

Token	operation
operand	Add to stack
operator	pop stack into n_1 pop stack into n_2 perform $n_3 = n_1$ operator n_2 push n_3

pop stack to obtain result.

+ − * $ 4 2 3 3 / / 8 4 + 1 1

Token	stack
1	1
1	1, 1
+	2
4	2, 4
8	2, 4, 8
/	2, 2
/	1
3	1, 3
3	1, 3, 3
2	1, 3, 3, 2
4	1, 3, 3, 2, 4
$	1, 3, ~~16~~ 3, 16
*	1, 3, 48
−	1, 45
+	$\boxed{46}$

$n_1 = 1$
$n_2 = 1$

$n_3 = n_1 / n_2$

Algorithm - 5

post to pre algorithm :-

scan from L to R Repeat 1 - 2

Token	operation
Operand	push to stack
Operator	pop stack into s_1 pop stack into s_2 Concatenate operator s_2 s_1 push the result on stack

pop stack to obtain the result.

AB$C*D-EF/GH+/+

Token	stack
A	A
B	A,B
$	$\$\overset{s_2}{A}\overset{s_1}{B}$
C	$AB,C
*	*$ABC
D	*$ABC,D
-	-*$ABCD
E	-*$ABCD,E
F	-*$ABCD,E,F
/	-*$ABCD,/EF
G	-*$ABCD,/EF,G
H	-*$ABCD,/EF,G,H
+	-*$ABCD,/EF,+GH
/	-*$ABCD,//EF+GH
+	+-*$ABCD//EF+GH

$ A [illegible] B [illegible] / [illegible] +//GH

$s_1 = B$
$s_2 = A$
$AB
$AB

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

ABC * DEF$ / G * - H * +

Token	Expression
A	A
B	A, B
C	A, B, C
*	A, *BC
D	A, *BC, D
E	A, *BC, D, E
F	A, *BC, D, E, F
$	A, *BC, D, $EF
/	A, *BC, /D$EF
G	A, *BC, /D$EF, G
*	A, *BC, */D$EFG
−	A, −*BC*/D$EFG
H	A, −~~*BC~~*/D$EFG, H
*	A, *−*BC*/D$EFGH
+	$\boxed{+A*-*BC*/D\$EFGH}$

Conversion of Infix to prefix.

```c
#include<stdio.h>
#include<conio.h>
#define MAX 50
#include<string.h>

char s[MAX], infix[MAX], pre[MAX], ele, ch, x, t;
int i=0, j=0, top=-1;
void push(char ch)
{ s[++top] = ch;
}
char pop()
{
  return (s[top--]);
}
int priority(char ch)
{ if (ch=='^')
    return 4;
  if(ch=='*' || ch=='/' || ch=='%')
    return 3;
  if(ch=='+' || ch=='-')
    return 2;
  else
    return 0;
}
void check()
{ while(priority(t)<priority(s[top]))
  pre[j++] = pop();
}
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
main()
{
  clrscr();
  printf(" Enter the infix expression:");
  scanf("%s" infix);
  push('#');
  strrev(infix);
  while (infix[i]!='\0')
   {
    t = infix[i];
    if (isalpha(t))
      ~~post~~ pre[j++] = t;
     else
      {
       if (t=='+' || t=='-' || t=='*' ||
           t=='/' || t=='%' || t=='^'
           || t=='(' || t==')')

      switch(t)
      {
       case ')' :  push(t); break;
       case '+' :
       case '-' :
       case '%' :
       case '/' :
       case '*' :
       case '^' :  check();
                   push(t);
                   break;
       case '(' :  do
                   {
                      x = pop();
                      pre[j++] = x;
                   }
                   while (x!='(');
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
    j=j-1;
    break;
   }
 }
 i=i+1;
}
while(s[top]!='#')
 {
  pre[j++]=pop();
 }
 pre[j]='\0';
 strrev(pre);
 strrev(infix);
 printf("The prefix Notation of given expression
                                  is %s", pre);
 getch();
}
```


program to evaluation of prefix.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<math.h>
#define MAX 50
char s[MAX],ch;
float st[MAX], op1, op2, op, val, temp, res;
int i=0, j=0, top=-1;
float pop()
 { return (s[top--]);
 }
void push(float val)
 { top++;
   s[top]=val;
 }
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
float operate(float op1, float op2, char ch)
 {
  switch(ch)
   {
     case '+' : temp = op1 + op2;
                break;
     case '-' : temp = op1 - op2;
                break;
     case '*' : temp = op1 * op1;
                break;
     case '/' : temp = op1 / op2;
                break;
     case '^' : temp = pow(op1, (int)op2);
                break;
   }
   return temp;
 }
main()
 {
  clrscr();
  printf("Enter the infix expression :");
  scanf("%s", s);
  strcpy(st, strrev(s));
  while(st[i] != '\0');
   {
     ch = st[i];
      if(isalpha(ch))
       {
         printf("Enter the value for %c = ", ch);
         scanf("%f", &val);
         push(val);
       }
     if(ch == '+' || ch == '-' || ch == '*' || ch == "%"
                || ch == '/' || ch == '^')
      {
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
        op2 = pop();
        op1 = pop();
        res = operate (op1, op2, ch);
         push(res);
       }
   }
   i = i+1;
  }
   temp = pop();
   printf ("The simplified answer for %s =", st);
   printf ("%f", temp);
   getch();
  }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

5/12/2016

Algorithm - 6

post to infix conversion :-

Scan from L to R Repeat 1 - 2

Token	stack
operand	push to stack
operator	pop stack into S1 pop stack into S2 Concatenate S2 operator S1 push result on stack

pop stack to obtain result.

AB$C*D-EF/GH+/+

Token	stack
A	A
B	A, B
$	A$B
C	A$B, C
*	A$B*C
D	A$B*C, D
−	A$B*C−D
E	A$B*C−D, E
F	A$B*C−D, E, F
/	A$B*C−D, E/F
G	A$B*C−D, E/F, G
H	A$B*C−D, E/F, G, H
+	A$B*C−D, E/F, G+H
/	A$B*C−D, E/F/G+H
+	A$B*C−D+E/F/G+H

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

ABC * DEF$ / G * - H * +

Token	Stack
A	A
B	A, B
C	A, B, C
*	A, B*C
D	A, B*C, D
E	A, B*C, D, E
F	A, B*C, D, E, F
$	A, B*C, D, E$F
/	A, B*C, D/E$F
G	A, B*C, D/E$F, G
*	A, B*C, D/E$F*G
-	A, B*C-D/E$F*G
H	A, B*C-D/E$F*G, H
*	A, B*C-D/E$F*G*H
+	$\boxed{\text{A+B*C-D/E\$F*G*H}}$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Algorithm - 7

pre to post algorithm :-

Scan from R to L repeat 1 - 2

Token	operation
operand	push to stack
operator	pop stack into S_1 pop stack into S_2 Concatenate S_1 S_2 operator push result on stack.

pop stack to obtain result.

+ - * $ A B C D / / E F + G H

Token	stack
H	H
G	H, G
+	G H +
F	G H +, F
E	G H +, F, E
/	G H +, ~~E~~ E F /
/	E F / G H + /
D	E F / G H + /, D
C	E F / G H + /, D, C
B	E F / G H + /, D, C, B
A	E F / G H + /, D, C, B, A
$	E F / G H + /, D, C, A B $
*	E F / G H + /, D, A B $ C *
-	E F / G H + /, A B $ C * D -
+	$\boxed{\text{A B \$ C * D - E F / G H + / +}}$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

+ A * - * B C * / D $ E F G H

Token	Stack
H	H
G	H, G
F	H, G, F
E	H, G, F, E
$	H, G, E ~~$~~ F $
D	H, G, ~~E$F~~ EF$, D
/	H, G, D ~~$~~ E $ F /
*	H, ~~D/E$FGDEF$/~~ DEF $ / G *
C	H, DEF$ / G *, C
B	H, DEF $ / G *, C, B
*	H, DEF $ / G *, BC *
-	H, BC * DEF $ / G * -
*	BC * DEF $ / G * - H *
A	BC * DEF $ / G * - H *, A
+	$\boxed{\text{ABC * DEF \$ / G * - H * +}}$

Algorithm - 8

Pre to Infix conversion :→

Scan from R to L Repeat step 1 - 2

Token	operation.
Operand	Push to stack
Operator	Pop stack into S_1 Pop stack into S_2 Concatenate S_1 operator S_2 Push result on stack

+ - * $ A B C D / / E F + G H

Token	Stack
H	H
G	H, G
+	G + H
F	G + H, F
E	G + H, F, E
/	G + H, E / F
/	E / F / G + H
D	E / F / G + H, D
C	E / F / G + H, D, C
B	E / F / G + H, D, C, B
A	E / F / G + H, D, C, B, A
$	E / F / G + H, D, C, A $ B
*	E / F / G + H, D, A $ B * C
-	E / F / G + H, A $ B * C - D
+	$\boxed{A \$ B * C - D + E / F / G + H}$

Conversion	scan order	Token	operation
In - Post	L - R	operand	Add to expression ~~priority~~
		operator	priority
In - pre	R - L	(	Add to stack / Delete
		)	Add to stack / Delete
Post - pre	L - R	operand	push to stack
Post - In	L - R	operator	pop stack into S_1 pop stack into S_2
Pre - post	R - L		construct
Pre - In	R - L		Expression Push expression on stack

Dynamic Memory allocations :→

→ As we know that an array is an static memory allocation at the time of compilation we need to define the size once the size has been defined it is never possible either to increase or decrease the size.

→ To overcome this problem we use dynamic memory allocation.

→ The dynamic memory allocation will takes place by dynamic functions like, malloc (), calloc(), realloc() and free().

→ These dynamic memory functions allocate the memory from the heap area.

→ when we are allocating the memory by using this functions we need to perform explicit typecasting, since this function return type is void *.

malloc() :→

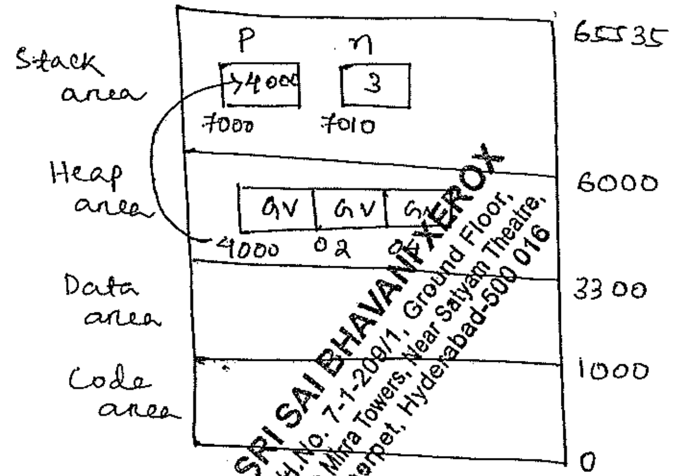
Used to allocate required number of bytes in memory at run time.

→ It takes one argument i.e size in bytes to be allocated.

Syntax :-

ptr_Variable = (type_cast *) malloc(sizeof(size));

ex:-
```
int *p;
int n;
P = (int *) malloc (n * sizeof(int));
```



6/12/2016

What is a linked list?

A linked list is a data structure which is collection of 0 or more nodes, where each node has some information.

→ A node will be heterogeneous type which can store any type of data. Basically each node is divided into two parts, the 1st part contains the information of the element and the 2nd part contains address of the next node which is called as Link.

→ The node will be created with the structures concept.

How to create a node?

Syntax:

```
struct node
{
    type 1  info;
    type 2  *link;
};
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Note: Here struct is a keyword and node is tagname. The node consists of two fields info and link.

Info: since it contains the information, it can be of type int, float, char, double.

eg: int info;
char info;

Link :

It contains address of the next node, so, link field must be pointer to a node which can be declared as shown below.

```
struct node * link;
```

→ The link field must be same type since we are holding the address of a node of the same type. The type of structures is called as self refferential structures.

→ By this structures we can hold the address of a node of same type.

```
int a;
int *p;
P = &a;
float s;
P = &s; X
```

The structure defination of a node along with declaration can be done as follows.

```
struct node
{
    int info;
    struct node * link;
};
typedef struct node * NODE;
```

```
main()
{
typedef int number;
typedef char * string;

number a=45;
string s1, s2="abc";
```

A pointer variable first can be declared as shown below

Method 1: struct node * first;

or

Method 2: NODE first;

The info and link field can be accessed using the following notations,

Notation 1: Using * operator for Method 1

(* first).info

(* first).link

Notation 2: Using → operator for method 2.

first → info

first → link

Applications of Linked list :→

→ Linked list concepts are useful to model many different extract data types such as ques, stack and trees.

Advantage of Linked list :→

→ A linked list is a dynamic data structure and therefore the size of the linked list can be increased or decreased during the execution of program.

→ A linked list ~~doesnot~~ does not require any extra space, so it doesnot waste extra memory.

→ It provides flexibility in rearranging the items very efficiently.

→ The ~~utilis~~ limitation of linked list is that it consumes more memory space compared to array. since each node must also contain the address of the next item.

→ It is time consuming and burden process for checking the items in the list.

Types of Linked List :→

Depends on the applications we use different types of linked list like

1. Linear Singly Linked list
2. Circular singly Linked List
3. Circular Doubly Linked list
4. Doubly Linked List.

→ The general properties on a linked list successive elements are connected by pointers.

→ Basically the last element points to null.

The general operations of linked list are

1. creation
2. Insertion
3. Deletion
4. Traversing
5. searching
6. concatenation
7. modification etc.

Linear singly Linked List :→

By using linear singly linked list we can implement linear data structure operations.

program on single linked list

```
Struct emp
 {
   int eno;
   char ename[20];
   struct emp *link;
 } *start;
```

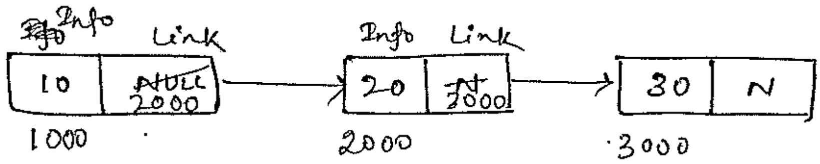
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

7/12/2016

→ In a single linked list a node has minimum 2 fields information field and the link field.

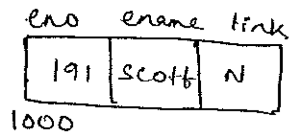
→ Every node will link to the other node by placing the address of the next node in the previous node of the link field.

→ The last node of the link field indecating by NULL, that indecates end of the node.

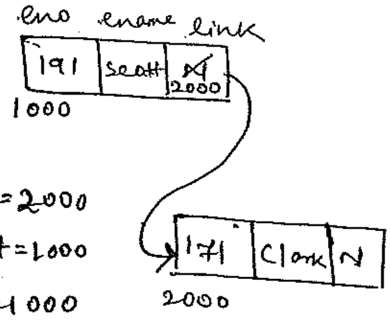


```c
#include<stdio.h>
struct emp
 {
    int eno;
    Char ename[20];
    Struct emp *link;
 } *start;
Creation ()
{
 struct emp *tmp, *q;
 tmp = (struct emp *) malloc (sizeof (struct emp));
 printf ("Enter the eno: ");
 scanf ("%d", &tmp -> eno);
 printf ("Enter the Ename: ");
 scanf ("%s", &tmp -> ename);
 tmp -> Link = NULL;
 if (start == NULL)
   start = tmp;
 else
  {
   q = start;
   while (q -> link != NULL)
    {
     q = q -> link;
    }
   q -> link = tmp
  }
} // end of creation.
```


tmp = 1000
start = ~~N~~ 1000

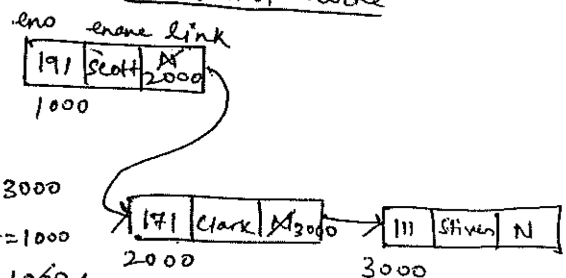


for second node



tmp = 2000
start = 1000
q = 1000

for third node

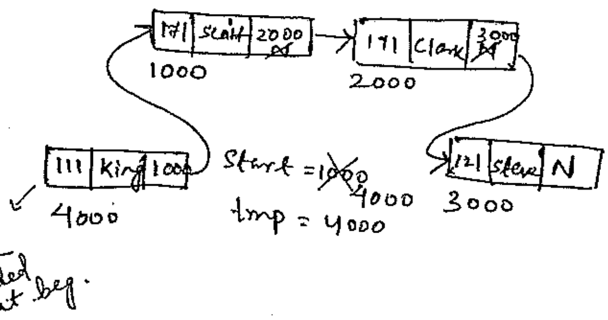


tmp = 3000
start = 1000
q = ~~1000~~ ~~2000~~ 2000

```
void display()
{
  struct emp *q;
  q = start;
  while (q != NULL)
  {
   printf("| %d %s
  if (q == NULL)
     printf("\n List is empty ");
  else
   {
     printf("\n The data in the list ...\n");
   while (q != NULL)
    {
    printf("| %d   %s    (%d) | %d |--->", q->eno, q->ename,
                                                q, q->link);

     q = q->link;
    }
   }
 }
addatbeg()
{
 struct emp *tmp;
  tmp = (struct emp *)malloc(sizeof(struct emp));
  printf("Enter the eno: ");
  scanf("%d", &tmp->eno);
  printf("Enter the Ename: ");
  scanf("%d", &tmp->ename);
   tmp->link = start;
   start = tmp;
 }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016



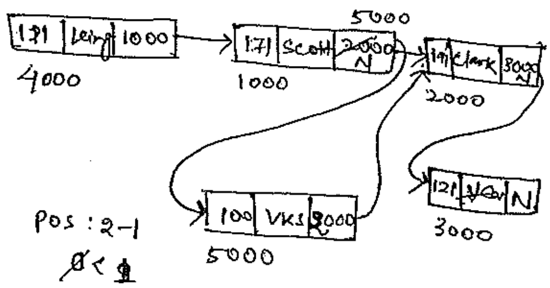
```
addafter(int pos)  // pos=6
{
 struct emp *q, *tmp.
 int i;
 q=start;
   for(i=0; i< pos-1; i++)
     {
       q=q->link;
        if(q==NULL)
         {
          printf("There are less than %d elements", pos);
          return;
         }
     } // for

 tmp = (struct emp *) malloc(sizeof(struct emp));
 printf("Enter the eno :");
 scanf("%d", &tmp->eno);
 printf("Enter the Ename:");
 scanf("%s", &tmp->ename);
 tmp->link = q->link;
  q->link = tmp;
}
```

pos = 6

q=4000	$i = \cancel{0} < 5$
1000	$\cancel{1} < 5$
2000	2<5
3000	3<5
N	

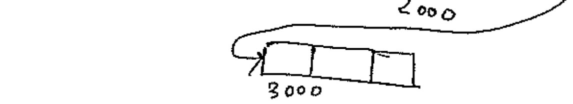
for pos ≥ 2



pos : 2-1
$\cancel{0} < 1$
1 < 1

q=1000
tmp=5000

```
del (int data)
{
 struct emp *tmp, *q;
  if(start->eno == data)
   {
    tmp = start;
    start = start->link; // first element deleted.
    free (tmp);
    return;
   }
```



tmp = 4000
start = 1000
free(4000)

```c
q = start;
while (q->link != NULL)
  {
    if (q->link->eno == data) // element deleted in
                                        between
    {
      tmp = q->link;
      q->link = tmp->link;
      free(tmp);
      return;
    }
    q = q->link;
  }
  printf("Element %d not found \n", data);
}
```

```c
void delete[illegible]()
{
   [illegible] emp * tmp;
   while ( [illegible] )
    {
      tmp = start;
      [illegible] link;
    } free(tmp);
}  // ( Successfully
        deleted
        all
```

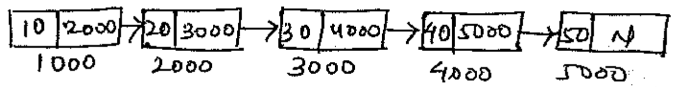
```c
main()
{
  int choice, i, n, pos, ele
  start = NULL;
  clrscr();
  while(1)
  {
   printf("\n 1. create a list \n");
   printf("\n 2. Display \n");
   printf("\n 3. add at beg \n");
   printf("\n 4. addafter \n");
   printf("\n 5. deletation \n");
   printf("\n 6. Exit ");
   printf("\n Enter the choice:");
   scanf("%d", &choice);
   switch (choice)
   {
     case 1: printf("\n How many nodes do you want");
             scanf("%d", &n);
             for(i=0; i<n; i++)
             creation();
             break;
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
case 2: display();
        break;
case 3: addatbeg();
        break;
case 4: printf("Enter the position:");
        scanf("%d", &pos);
        addafter(pos);
        break;
case 5: printf("Enter the empno to delete :");
        scanf("%d", &ele);
        del(ele);
        break;
case 6: Exit(0);
  } //end of switch.
 } // end of while.
 getch();
} // end of main.
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```c
#include<stdio.h>
struct node
{
  int info;
  struct node *next;
} *start;
Void create()
{
  struct node *new, *t;
  new = (struct node *) malloc(sizeof (struct node));
  printf ("Enter the information :");
  scanf ("%d", &new->info);
  new->next=NULL;
    if ( start == NULL)
      start = new;
    else
    {
      t = start;
      while (t->next != NULL)
      {
        t = t->next;
      }
      t->next = new;
    }
}
```

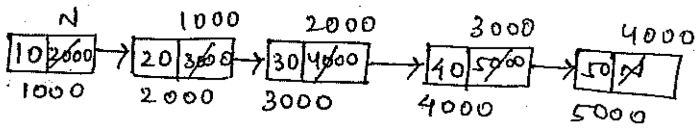
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```c
Void display()
{
  struct node *q;
  q = start;
  if (q == NULL)
    {
      printf ("The list is empty");
      return;
    }
```

```
while (q != NULL)
  { printf ("%d", q->info);
    q = q->next;
  }
}
```



```
struct node * reverse (struct node * P)  // P=1000
{
  struct node *q, *r;
  q = NULL;
  while (P)  //1000
  {
    r = P->next;
    P->next = q;
    q = P;
    P = r;
  }
  return q;
}
```

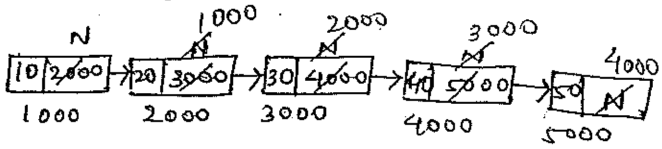


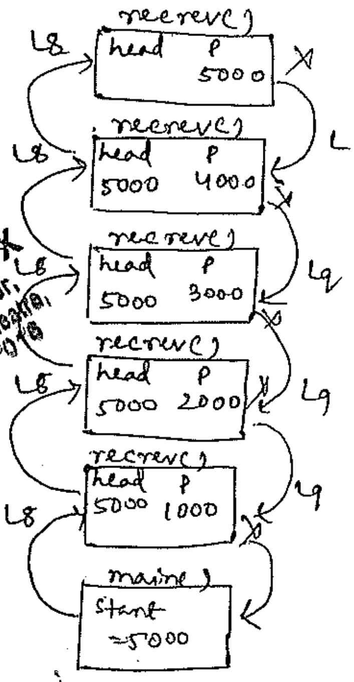
P = ~~1000~~ ~~3000~~ ~~4000~~ ~~5000~~ N

q = ~~N~~ ~~1000~~ ~~2000~~ ~~3000~~ ~~4000~~ 5000

r = ~~2000~~ ~~3000~~ ~~4000~~ ~~5000~~ N

```
1  struct node * recrev (struct node * P)
2    {
3      struct node *head;
4      if (!P)
5      return NULL;
6      if (P -> next)
7        {
8          head = recrev (P->next);
9          P -> next -> next = P;
10         P -> next = NULL;
11         return head;
12       }
13     else
14       return P;
15   }
```



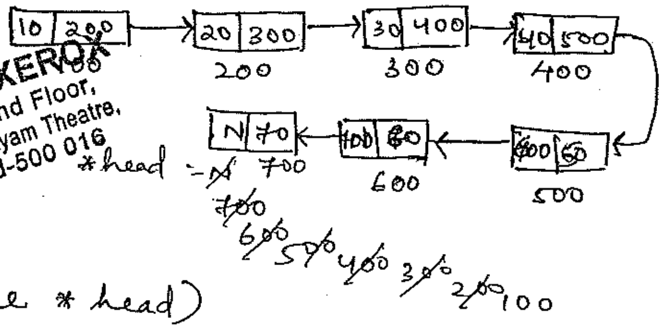


```
main()
{ int choice, i, n,
   start = NULL;
   clrscr();
   while(1)

   { printf ("\n 1. create a list \n");
     printf ("\n 2. Display \n");
     printf ("\n 3. reverse \n");
     printf ("\n 4. rec rev \n");
     printf ("\n 5. Exit \n");
     switch (choice)
      {
      case 1: printf ("\n How many nodes do you want");
              scanf (" %d", &n);
              for(i=0 ; i<n; i++)
              creation( );
              break;
      case 2: display();
              break;
      case 3: start = reverse (start);
              break;
      case 4: struct recrev(start);
              break;
      case 5: exit(0);
      }
   }
   getch();
}
```

Q) write a program for single linked list and display the nth node from last ?


```
Struct Node
 {
   int data ;
   struct Node *next ;
 };
// inserting node at starting for simplicity.
CreateNode (struct Node ** head , int data)
 {
   struct Node *temp =(struct Node *) malloc (sizeof
                                         (struct Node));
   temp -> data = data ;
   temp -> next = *head ;
   *head = temp ;
 }
```




```
Void display (struct Node * head)
 {
   while (head)
    {
      printf ("%d", head -> data);
      head = head -> next ;
    }
 }
// Finding the Nth in singly linked list from first
 Void findNthNode (struct Node *head , int n)// 100 3
 { int count = 1;
   struct Node * temp = head ;
   if (head = = NULL)
    {
```

```
printf ("List is empty \n");
 return;
}
while (count < n && temp)
 {
  temp = temp -> next;
  count ++;
 }
 if (count == n)
 {
  printf (" The required Node at the location %d
                is %d \n", n, temp -> data);
  return;
 }
else
  printf ("Less No of Nodes are present in the
                linked list");
 }
```



```
// Finding the Nth Node in single linked list from last
void FindNthNodefromLast (struct Node *head, int n)
 {                                                    //1003
  int count;                                          100 5
  struct Node *temp = head, *Nthnode = NULL;
  if (head == NULL)
   {
    printf ("List is empty");
    return;
   }
  for (count = 1; count < n; count ++) //3
   {
    if (temp) //100 200
     temp = temp -> next; //300
   }
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
while (temp)
{ if (Nthnode == NULL)
    Nthnode = head ; //100
  else
    Nthnode = Nthnode -> next ;
  temp = temp -> next ;
}
if (Nthnode)
  printf ("The required Node at the location %d is
                 %d \n", n, Nthnode -> data);
else
  printf ("Less No: of Nodes are present in linked
                                          list");
}

main()
{ struct Node *head = NULL;
  int n;
  CreateNode (&head ,70);            head=100
  Create Node (&head , 60);           n=5
  CreateNode (&head , 50);
  CreateNode (&head ,40);
  CreateNode (&head, 30);
  CreateNode (&head, 20);
  CreateNode (&head , 10);
  printf ("Elements in the linked list ..\n");
  display (head);
  printf ("\n");
  printf ("Enter the position of required node: \n");
  scanf ("%d", &n);
  findNthnode (head , n);
  printf ("\n");
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
printf("Enter the position of required node to find from
                                                   last:\n");
scanf("%d", &n);
findNthNodefromLast(head, n);
return 0;
}
```


Write a program to swapping the kth node data from the begining and kth node data from last in the single linked list.

```
Void swap(struct Node *head, int k)
{ int count, p;
  Struct Node *tmp1 = head, *tmp2 = head, *tmp3 =
                                                NULL;
  if(head == NULL)
  { printf("List is empty");
    return;
  }
 for(count = 1; count < K; count++)
  { if (tmp1)
       tmp1 = tmp1 -> next;
  }
 tmp2 = tmp1;
  while (tmp2)
   {
     if (tmp3 == NULL)
       tmp3 = head;
     else
       tmp3 = tmp3 -> next;
       tmp2 = tmp2 -> next;
   }
```

```
    P = tmp1 -> data;
    tmp1 -> data = tmp3 -> data;
    tmp3 -> data = P;
}
```


13/12/2016

```
void swap(struct node **head, int K)
{
  struct node *ptr, *kthnodefrombeg, *kthnodefromlst;
  int count=1, temp;
  kthnodefromlst = kthnodefrombeg = *head;   // 100
  if (K==0)
  {
  printf("There is no nodes to be display\n");
  exit(0);
  }
// Finding the kth node from begining
while (kthnodefrombeg && count<k)   // 300
  {
  count ++;
  kthnodefrombeg = kthnodefrombeg -> next;
  }
// if the while loop terminates due to NULL then
  if (!kthnodefrombeg)
  {
  printf("There are less No: of Nodes in the list \n");
  exit(0);
  }
ptr = kthnodefrombeg -> next;
// Now move 2 pointer at a time if ptr reaches last
  position then kthnodefromlst points to the kth
  node from the last
```

```
while (ptr)
 { ptr = ptr->next;
   kthnodefromlst = kthnodefromlst ->next;
 }
// swapping kthnodefrombeg and kthnodefromlst using
temporary variables temp
// here we swapping the data only.

  temp = kthnodefrombeg -> data;
  kthnodefrombeg -> data = kthnodefromlst -> data;
  kthnodefromlst -> data = temp;
}
 main()
 {
  struct Node *head = NULL;
   int K;
   clrscr();
   createNode (&head, 70);
   createNode (&head, 60);
   createNode (&head, 50);
   createNode (&head, 40);
   createNode (&head, 30);
   createNode (&head, 20);
   createNode (&head, 10);
   printf (" Enter the K value :");
   scanf ("%d", &K);
   printf ("\n Before swapping \n");
   display (head);
   printf ("\n After swapping \n");
   swap (&head, K);
   display (head);
   return 0;
 }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Enter the k value : 3

Before swapping

10 20 30 40 50 60 70

After swapping

10 20 50 40 30 60 70

Q) Given a null terminated linked list, rearrange its nodes into two lists (first node, 3rd node, 5th node) and (2nd node & 4th node, 6th node)

Sample I/O formats

Enter the input linked list values

Enter the value

5

Do you want to add another node [Y/N]

Y

Enter the value

8

10

3

90

The elements in the linked list are: 5 8 10 3 90

The elements in the 1st linked list are; 5 10 90

The elements in the 2nd linked list are: 8 3

```
struct node
{
    int data;
    struct node *next;
} *head = NULL;
struct node *headoflst1 = NULL;
struct node *headoflst2 = NULL;
void arrangelist( )
{
    struct node *temp = head, *list1 = NULL, *list2 = NULL;
    struct node *ptr = 
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
struct node
 {
   int data;
   struct node * next;
 } *head = NULL;
creation()
 {
  char ch;
  do
  {
   struct node *tmp , *q;
   tmp = (struct node *)malloc(sizeof (struct node));
   printf (" Enter the data :");
   scanf ("%d", &tmp -> data);
   tmp -> next = NULL;
    if (head == NULL)
      head = tmp;
    else
    { q = head;
      while (q -> next != NULL)
        q = q -> next;
        q -> next = tmp;
     }
    Printf (" Do you want to add another node:\n");
      ch = getche();
   } while (ch != 'n');
  }
 Void display(struct node * tmp)
   {
     while ( tmp != NULL)
     {
       printf (" %d", tmp -> data);
        tmp = tmp -> next;
     }
   }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
Struct ~~tmp~~ node *headoflst1 = NULL;
struct node * headoflst2 = NULL;
Void Rearrange()
 {
  struct node *temp = head, *list1 = NULL, *list2 = NULL;
  struct node *ptr;
  int i=1;
  while (temp != NULL)
   {
     if(i%2==1)
      {
       ptr = (struct node *)malloc(sizeof(struct node));
       ptr->data = temp->data;
       ptr->next = NULL;
       if (headoflst1==NULL)
        {
          headoflst1 = ptr;
        }
      else
          list1->next = ptr;
          list1 = ptr;
      }
    else
      {
       ptr = (struct node*)malloc(sizeof (struct node));
        ptr->data = temp->data;
        ptr->next = NULL;
         if (headoflst2 = =NULL)
           headoflst2 = ptr;
       else
         list2->next = ptr;
          list2 = ptr;
       }
      temp = temp->next;
  }  }  i++;
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
main()
{
    struct node * head = NULL;
    printf("\n Enter the Input linked list values : ");
    Creation( );
    printf ("Before arrange list is : ");
    display (head);
    Rearrange ( );
    printf(" The elements in the 1st linked list are : ");
    display (headoflst2);
    printf (" The elements in the 2nd linked list are: ");
    display ( headoflst1);
    return 0;
}
```


Write a program to delete the node without using head pointer in a single linked list.

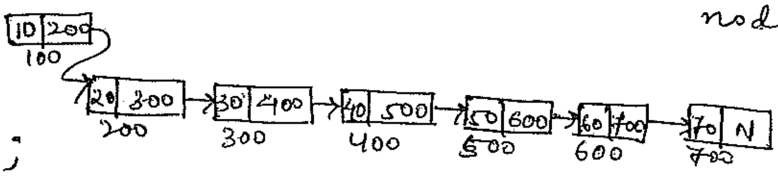
for this problem we need to know the actual address of the required node, hence we need to save the address by using a global variable. In this program suppose i am saving the 3rd node address from last.

```
void delete(struct node * nodetodel)
{
    struct node * nextnode;
    if (nodetodel = = NULL)
        return;
    nextnode = nodetodel -> next;
    nodetodel -> next = nextnode -> next;
    free(nextnode);
    return;
}
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

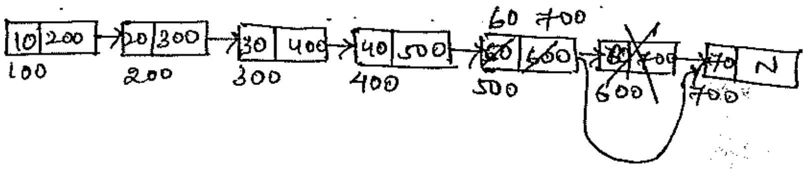
14/12/2016

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
 {
   int data;
   struct node * next;
 };
struct node *ptr1;
int count;
// create a Node
void createnode(struct node **head, int data)
 {
 struct node * newnode = (struct node *)malloc(sizeof(struct
                                                       node));
  count ++;
  if (count == 3)
     ptr1 = newnode;
     newnode -> data = data;
     newnode -> next = *head;
     *head = newnode;
 }
void display(struct node *head)
 { if (head == NULL)
    {
     printf("List is empty");
     return;
    }
   while(head)
    {
     printf("%d\t", head -> data);
     head = head -> next;
    }
 }
```



// Deleting the node without using head pointer.
// we need to copy the next node information into current
// node and we are actually deleting the next node
// (since we have all the information of next node in
// current node)

```
void deletenode(struct node *ptr)        =500
  {
  // saving the next node address
   struct node *temp = ptr->next;
   ptr->data = ptr->next->data;
   //printf("%d\t", ptr->data);
     ptr->next = ptr->next->next;
   //printf("%d\t", ptr->next);
    free(temp);
  }
```



ptr = 500
tmp = 600

```
int main()
 {
   struct node *head = NULL;
   Createnode(&head, 70);
   createnode(&head, 60);
   Createnode(&head, 50);
   Createnode(&head, 40);
   Createnode(&head, 30);
   Createnode(&head, 20);
   Createnode(&head, 10);
   printf("Display the node before deletion\n");
   display(head);
 // calling the delete node function with saved argument
                                                   Ptr 1.
  deletenode(ptr1);
  printf("\n Display the node after deletion..\n");
  display(head);
  return 0;
 }
```

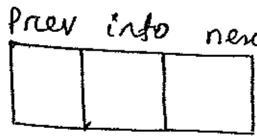

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

As in the single linked list the traversing of data will takes place only one way inorder to traversing the data ~~we~~ ~~as~~ in two ways i.e forward and backword, we use Double linked list.

→ The ~~many~~ major purpose of double linked list to implement non-linear data structures.

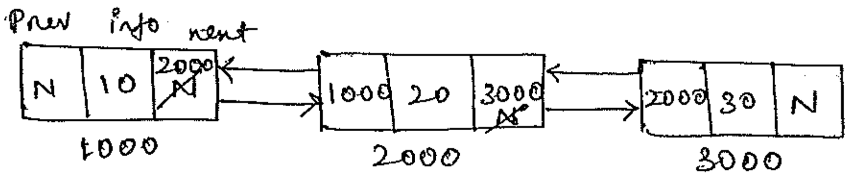
→ In double linked list a node has 3 fields.

(i) Information
(ii) previous field
(iii) Next field.



→ In double linked list the next field of the node contains next node address, the previous field of the node contains before node address. This makes double linked list.

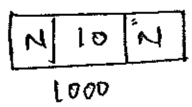
→ The first node of the previous field and next field of the last node contains NULL.



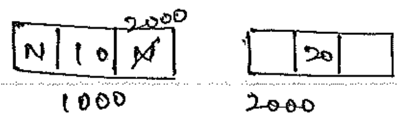

```
struct dll
{
   struct dll *prev;
   int info;
   struct dll *next;
} *start;

creation()
{
   struct dll *temp, *P

tmp = (struct dll *) malloc( sizeof (struct dll));
printf ("Enter the information");
scanf ("%d", &temp->info);
```



temp = 2000   start = 1000



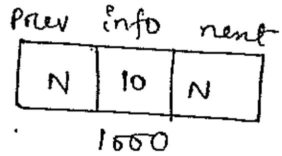
temp -> prev = NULL;
temp -> next = NULL;
if (start == NULL)
    start = temp;
else
{ ~~while (p -> next~~
    p = start;
    while (p -> next != NULL && p -> prev != NULL)
    {
        p = p -> next;
    }

    p -> next = temp;
    p -> prev = p;
}

temp -> next = NULL;
if (start == NULL)
    temp -> prev = NULL;
else
{
    p = start;
    while (p -> next != NULL)
    {
        p = p -> next;
    }
    p -> next = temp
    p.

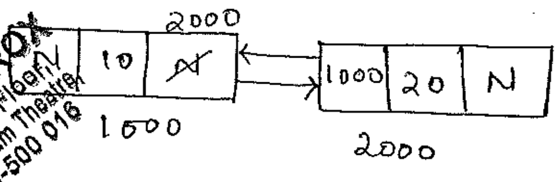
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```c
#include<stdio.h>
struct dll
{
 struct dll *prev;
  int info;
  struct dll *next;
 } *start;
creation()
{
 struct dll *new,*p;
new=(struct dll *)malloc(sizeof(struct dll));
printf("Enter the information");
scanf("%d",&new->info);
 new->next=NULL;
 if(start==NULL)
   {
     start=new;
    new->prev=NULL;
   }
 else
  {
   p=start;
  while(p->next!=NULL)
    {
     p=p->next;
    }
   p->next=new;
   new->prev=p;
   }
}
```



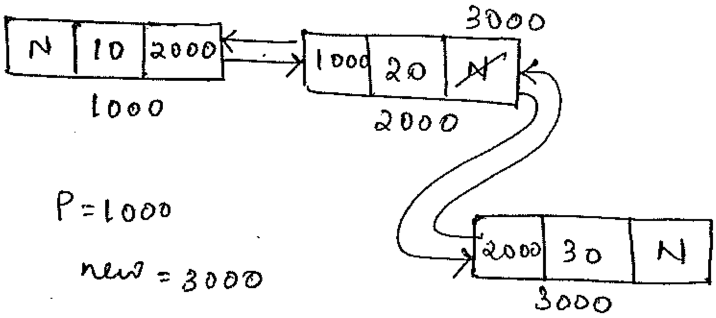
Start = 1000

for 2nd node



P = 1000

new = 2000



P=1000

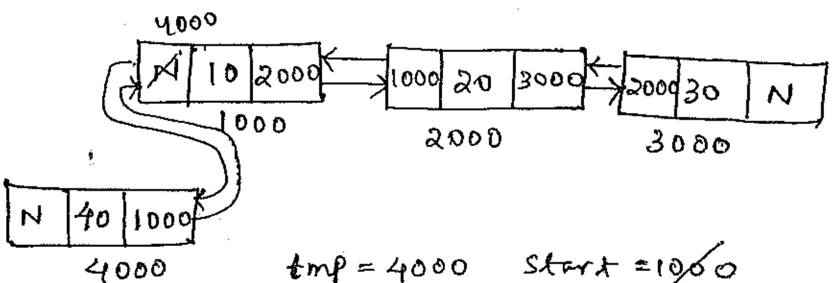
new = 3000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
void display()
{
  struct dll *q;
  q = start;
  if (q == NULL)
  printf("List is empty");
else
  {
  printf("Data in the list:");
  while(q != NULL)
    {
    printf("\t %u<-- %u--> %u", q->prev, q->info, q->next);
    q = q->next;
    }
  }
}
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
addatbeg()
 {
  struct dll *tmp;
 tmp = (struct dll *)malloc(sizeof(struct dll));
 printf("Enter the Information");
 scanf("%d", &tmp->info);
 tmp->next = start;
  start->prev = tmp;
  start = tmp;
  tmp->prev = NULL;
 }
```

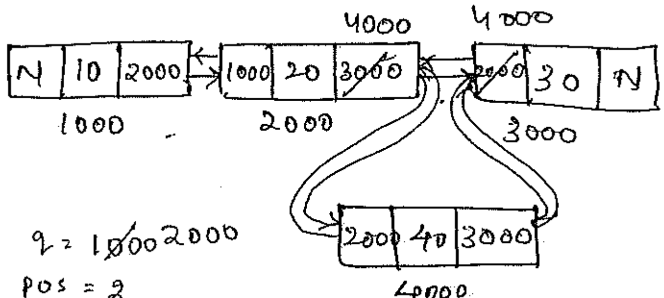


tmp = 4000   start = ~~1000~~ 4000

```
addafter(int pos)
  {
   struct dll *tmp, *q;
   int i;
   q = start;
```

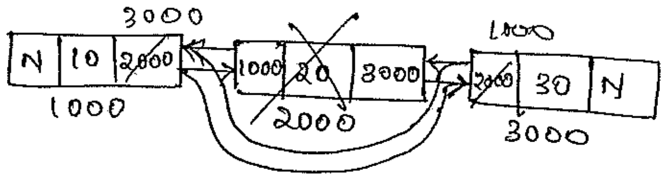
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
for( i=0 ; i< pos-1 ; i++)
  {
     q = q->next ;
     if (q == NULL)
       {
        printf (" There is less no of %d elements :", pos);
        return ;
       }
  }
   tmp = (struct dll *) malloc(sizeof(struct dll));
 printf ("Enter the information");
 scanf ("%d", &tmp->info);
   tmp ->next = q ->next ;
   q -> next = tmp ;
   tmp ->prev = q ;
 } tmp -> next -> prev = tmp ;
```



q = ~~1000~~ 2000
pos = 2
i=0   0 < 2-1 = 1
      1 < 1
tmp = 4000

```
del ( int  data)
{
 struct dll *tmp, *q;
   if (start -> info == data)
   {
    tmp = start ;
    start = start -> next ;
    start -> prev = NULL ;
    free (tmp) ;
    return ;
   }
  q = start ;
  while ( q -> next != NULL)
  { if (q->next->info == data)
    {
     tmp = q ->next ;
    q->next = tmp -> next ;
    tmp ->next -> prev = q ;
    free ( tmp ) ;
  } return ;
```



data = 20
q = 1000
tmp = 2000
free(2000)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
    q = q->next;
  }
  printf ("Element not found");
}

main()
{
 int choice, i, n, pos, ele;
Start = NULL;
 clrscr();
 while(1)
 {
  printf ("\n 1. create a list");
  printf ("\n 2. Display");
  printf ("\n 3. Add at begining");
  printf ("\n 4. Add after");
  printf ("\n 5. Delete");
  printf ("\n 6. Exit");
  printf ("Enter the choice");
  scanf ("%d", &choice);
  Switch (choice)
   {
   case 1 : printf ("\n How many nodes do you want :");
            scanf ("%d", &n);
            for(i=0 ; i<n ; i++)
             Creation();
             break;
  case 2 : display ( );
           break.
  case 3 : addatbeg ( );
           break;
  case 4 : printf ("Enter the position");
           scanf ("%d", & pos);
           addafter (pos).
           break;
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
case 5 : printf ("Enter the data to delete");
         scanf ("%d", &ele);
         del(ele);
         break;
case 6 : exit(0);
    }
 }
 getch();
}
```

```
Struct node
  {
    int data;
    struct node *prev;
    struct node * next;
  } *head, *last;

  head = NULL;
  last = NULL;

deletefrombeg()
 {
   struct node *todelete;

     todelete = head;
     head = head->next
     head->prev = NULL;
      free(todelete);
 }
deletefromend()
  { struct node * todelete;
       todelete = last;
       last = last->prev;
       last ->next = NULL;
  }    free(todelete);
```

```
deletefromN ( int pos)
{ struct node * tmp ;
    int i;
    tmp = head ;
    for( i=1; i<pos && tmp != NULL; i++)
     {
       tmp = tmp->next;
     }
    if (pos ==1)
    {
      deletefrombeg ( );
    }
  else
    if ( tmp == last)
     {
       deletefromend();
     }
   else if ( tmp != NULL)
    {
      tmp -> prev -> next = tmp -> next;
      tmp -> next -> prev = tmp -> prev;
       free( tmp);
    }
   else
      { Pf ("in valid position");
 }    }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

16/12/2016

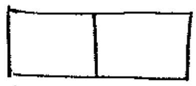

```
del ( int num)
{ struct node *tmp, *q;
  if (start -> info == num)
   {
     tmp = start;
     start = start ->next;   /* first element deleted */
     start ->prev = NULL;
     free(tmp);
     return;
   }
  q = start;
  while (q ->next ->next != NULL)
   {  if( q ->next ->info == num)
        { tmp = q -> next;
          q ->next = tmp ->next;   /* ~~last~~ element
          tmp ->next ->prev = q;      in between deleted */
          free(tmp);
          return;
        }
      q = q ->next;
   }
  if(q ->next -> info == num)
    {
      tmp = q ->next;   /* last element deleted */
      free(tmp);
      q ->next = NULL
      return;
    }
}
```

Circular singly linked list :→

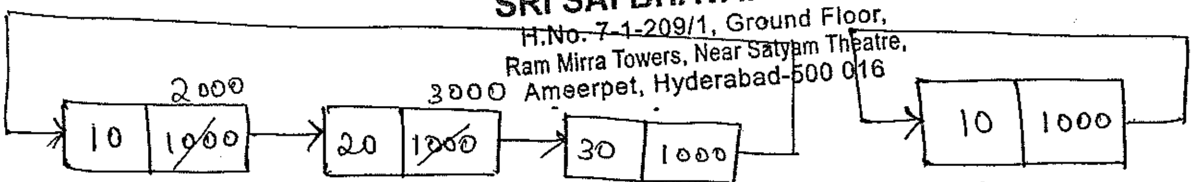
⇒ The drawback of doubly linked list is, there is a wastage of memory location of two fields, first node of the previous field and next field of last node.

⇒ To overcome this problem, we use circular linked list.

⇒ In circular singly linked list, a node has minimum two fields. (i) information
(ii) Next field.

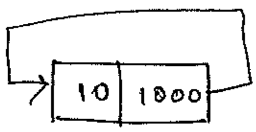


⇒ It is almost similar to single linked list, The next field of the last node contain the first node address.



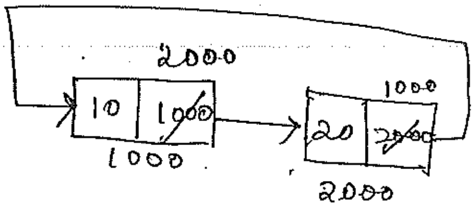

```
struct node
 {
   int info;
   struct node * link;
 } * last;

creation( )
 {
   struct node *new, *p;
  new = (struct node *)malloc(sizeof(struct node));
   printf("Enter the information");
   scanf("%d", &new->info);
  ~~new->link = new;~~
     if (last == NULL)
       {
         ~~last = new;~~   ~~last->link = new;~~
                          last = new;
                          new->link = new;
       }
```



else

```
    P = last;
    while (P -> link ! = last)
        {
           P = P -> link
        }
    P -> link = new
    new -> link = P;
  }
}
```



P=1000

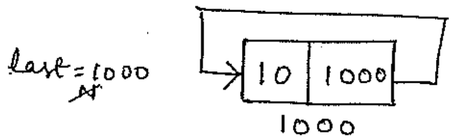
tmp = 2000

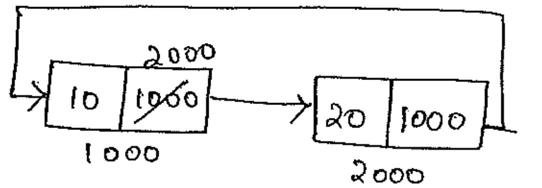

```
display()
{
   struct node * q;
     q = last;
   if (q == NULL)
     { printf("\n list is empty");
     }
  else
    while (q ! = last)
```

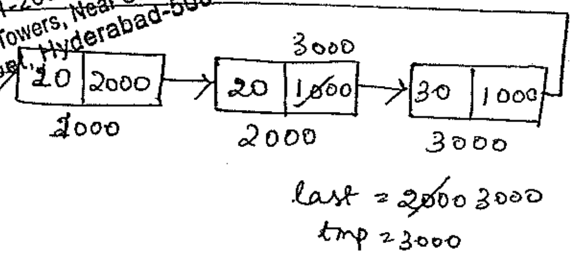
q = 2000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
struct node
 {
  struct node *link;
   int num;
 } *last;
creation(int num)
 {
   struct node *tmp, *q;
   tmp = (struct node *)malloc(sizeof(struct node));
   tmp => info = num;
    if (last == NULL)
      {
       last = tmp;
       tmp -> link = tmp;
      }
   else
    {
     tmp -> link = last -> link;
     last -> link = tmp;
      last = tmp;
    }
 display()
   {
    struct node *q;
      q = last -> link.
     if (last == NULL)
       {
        printf ("List is empty");
       }
     else
       while (q != last)
          {
           printf ("%d", q -> link);
             q = q -> link;
          }
         printf ("%d", last -> link);
   }
```

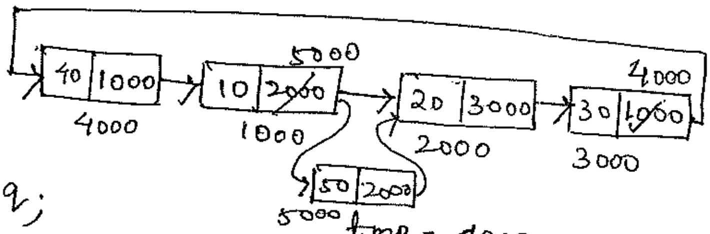






SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
addatbeg()
{
  struct node *tmp;
tmp = (struct node *)malloc(sizeof(struct node));
  tmp -> info = num;
  tmp -> link = last -> link;
  last -> link = tmp;
}
addafter(int pos, int num)
{
  struct node *tmp, *q;
  int i;
  q = last -> link;
  for(i=0; i< pos-1; i++)
         0 < 2-1
  {
    q=q -> link;
    if(q == last -> link)
    {
      printf("There are less no of nodes");
      return;
    }
  }
  tmp =(struct node *) malloc (sizeof (struct node));
    tmp -> info = num;
    tmp -> link = last -> link;        tmp -> link = q -> link;
    last -> link = tmp;                 q -> link = tmp;
    if(q == last)
       last = tmp;
}
```

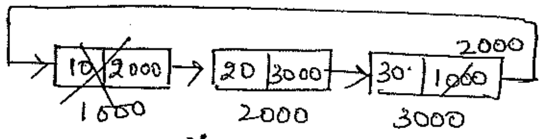


tmp = ~~4~~000
last = 3000

pos = 2
q = ~~4000~~ 1000
tmp = 5000

```
del(int num)
{
  struct node *tmp, *q;
   if (last -> link -> info == num)
    {
      tmp = last -> link;
     last -> link = tmp -> link;
        free(tmp);
        return;
    }
   q = last -> link;
   while(q! = last)
    {
      if(q -> link -> info == num)
        {
          tmp = q -> link;
           q -> link = tmp -> link;
           free(tmp);
           return;
        }
         q = q -> link;
    }
  if(last -> info == num)
   {
     tmp = last;
     last = tmp -> link -> link;
     last -> link = tmp -> link;
      free(tmp);
      return;
    }
}
```

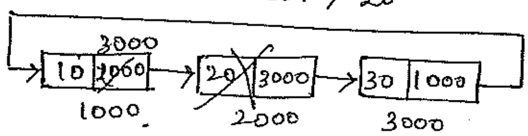
/* delete at beg */



tmp = 1000

last = 3000

/* delete in between */
del → 20

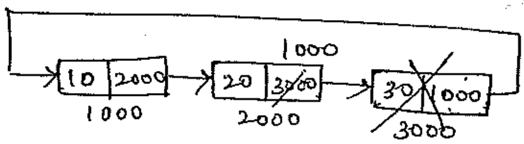


tmp..
↓
2000

q = 1000
last = 3000

free(2000)

/* delete last node */



last = ~~3000~~ 2000

tmp = 3000

free(3000)

17/12/2016


```
del (int num)
{ struct node * tmp, *q;
 if (last->link == last && last->info == num) // only one element deleted
  { tmp = last;
    last = NULL
    free (tmp); return;
  }
 q = last->link;
 if (q->link == num)
  {
    tmp = q;
    last->link = q->link;
    free (tmp);
    return;
  }
 while (q->link != last)
  { if (q->link->info == num) // element deleted in between
     { tmp = q->link;
       q->link = tmp->link;
       free (tmp);
       return;
     }
     q = q->link;
  }
 if (q->link->info == num) // last element deleted
  {                                  q->link = last
    tmp = q->link;
    q->link = last->link;
    free (tmp); last = q;
    return;
  }
  Printf ("Element %d not found \n", num);
}
```

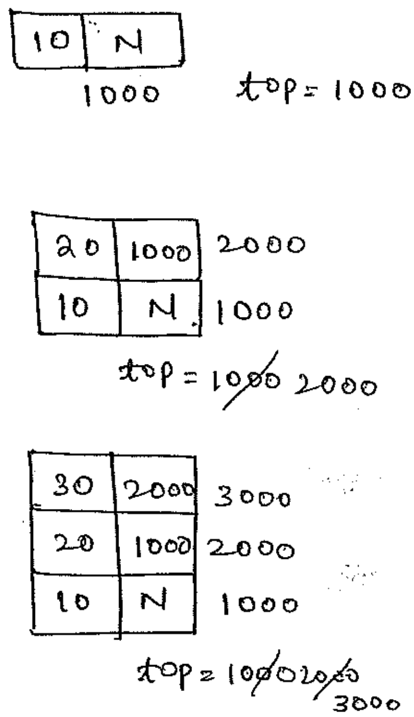
10

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Implement stack operations using linked list :-

```
#include<stdio.h>
#include<conio.h>
#define ISEMPTY top==NULL
struct stack
{
 int data;
 struct stack *next;
 };
typedef struct stack node;
node *top;
Void push(int item)
 {
  node *temp;
 temp = (node *) malloc( sizeof (node));
 temp ->data = item;
  temp -> next = top;
   top = temp;
  }
int pop ()
 { node * temp;
   int item;
   if (ISEMPTY)
    { printf ("\n stack is empty");
      return -1;
    }
   temp= top;
   item = temp -> data;
   top= top ->next;
   free(temp);
   return item;
 }
```



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
int peek()
  {   if (ISEMPTY)
        {
         printf ("stack is empty");
         return -1;
        }
     else
        return (top->data);
   }
int length()
 {
   node * a;
   int size = 0;
   for (a=top ; a!=NULL ; a=a->next)
       size++;
     return (size);
  }
Void display()
  {
   node * a;
   if (ISEMPTY)
     printf ("\n No item to display");
    else
     {
      printf ("\n The data is ");
      for (a=top; a!=NULL; a=a->next)
         printf ("\n %d", a->data);
     }
   }
main()
{
  int choice, item, i;
   clrscr();
   while (1)
    {
     printf ("\n\n 1. Push");
     printf ("\n 2. POP");
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
printf ("\n 3. Peek");
printf ("\n 4. isempty");
printf ("\n 5. Display data");
printf ("\n 6. Length");
printf ("\n 7. Exit");
printf (" Enter the choice :");
scanf ("%d", &choice);
switch (choice)
{
case 1 : printf ("Enter the item to push :");
         scanf ("%d", &item);
         push (item);
         break;
case 2 : if ((i=pop ())! =1)
         printf (" The deleted item is : %d", i);
         break;
case 3 : if ((i=peek ())! =-1)
         printf (" Top most element is : %d ", i);
         break;
case 4 : if (ISEMPTY)
         printf ("\n The stack is empty");
         else
         printf ("\n The stack is not empty");
         break;
case 5 : display ();
         break;
case 6 : printf ("\n The size is : %d ", Length ());
         break;
case 7 : exit(0)};
         default : printf (" \n Wrong choice ");
 } //switch
} //while
} // main
```

19/12/2016

TREES

→ The need of non-linear data-structures, improve the performance of an application by reducing the execution time.

→ Prevent loss of data

→ wastage of memory space is improved or optimise the performance.

→ The non-linear data structures are divided into two types (i) Trees
(ii) Graphs.

Defn of Tree:-

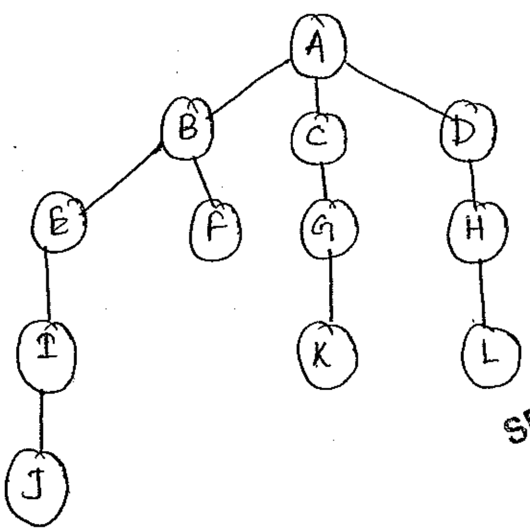
A Tree structure way of representing the hiearchial nature of a structure in a graphical form.

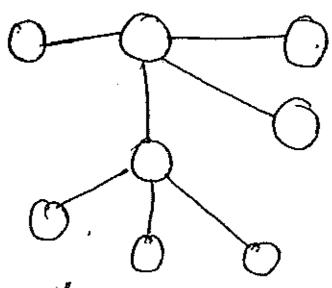
OR

It is defined as a tree is a finite set of nodes such that there is a specially designed node called root node. The remaining nodes are partitioned into different disjoints set $t_1, t_2, \ldots t_n$

⇒ A tree can be called as "connected acyclic graph".

→ It is a hiearchical graph.





SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

→ A node is nothing but collection of information along with its branches.

→ The above tree has a root node of 'A' and has the following terms

→ Degree of a node
→ Terminal non-terminal nodes
→ Parent Vs child
→ Siblings
→ Level of the node
→ Degree of the tree.
→ Height of the tree or depth of the tree.

Degree of a node :→

→ The number of sub-trees are childrens are called as degree of a node.
→ The degree of 'A' in the above tree (eg:- In above fig degree is 3)
→ The degree of tree is 3.

Terminal & non-Terminal nodes :-

→ A node with no children is called as terminal nodes. (leaf nodes) i.e whose degree is equivalent to zero.

examples :- J, F, K, L nodes in above diagram.

→ A node is said to be non-terminal whose degree is greater than zero.

example :- B, C, D, E, G, H, I.

Parent Vs child :-

→ If there is a branch from U → V then, U is called as Parent and V is called as child of U.

Siblings :-

→ childrens of the same parent are called as siblings.

ex :- B, C, D
E, F

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Level of the node :-

→ The level of the node indecates level no of the parent plus one.

level of A = 0 + 1 = 1.

level of B, C, D = 1 + 1 = 2

Degree of the tree :-

→ It is defined as the maximum degree in the tree.

ex:- In above fig. degree → 3.

Height or depth of the tree :-

→ The height of the tree is the largest level no in that tree.

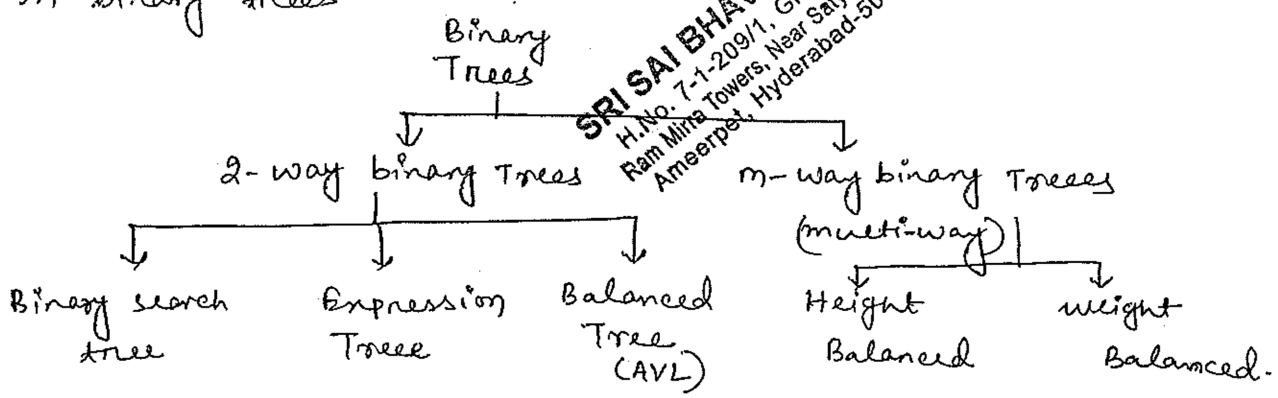
→ In the above tree the height is 5.

→ Basically trees are two types :-

(i) General Trees

(ii) Binary trees.

→ In binary trees



→ Difference Between Trees and Binary trees :→

1 → A binary tree can be empty where as a tree cannot

eg :- ○ → possible in binary
→ not possible in tree

2 → Each element in binary tree has exactly two sub trees (one or both of these sub trees may empty). Each element in a tree can have any number of sub trees.

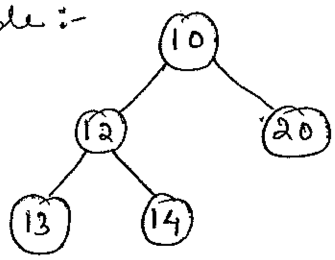
3) The sub tree of each element in a binary tree ~~but~~ are ordered, left and right sub trees.

4) The sub-trees in a tree are unordered.

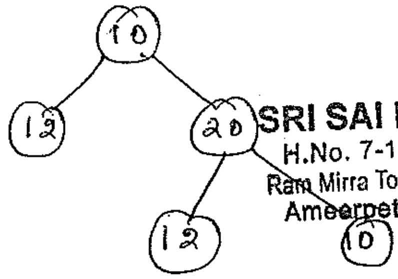
Strictly Binary Tree :-

If the out degree of every node in a tree is either 0 or 2, then the tree is said to be strictly binary tree.

Example :-



OR



Applications of Binary Trees :-

→ Expression trees are used in compilers.

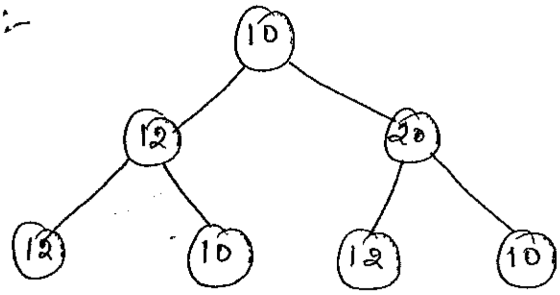
→ Huffman coding trees that are used in data compression algorithm.

→ Binary search trees which supports search, insertion and deletion on a collection of items in O(logn) (avg) etc.

Complete Binary Tree :-

In a complete binary tree, if there are 'n' nodes are existed then all the nodes must be consecutive (sequence) (for every $2i+1$ there must be exist $2i$)

example :-



Representation of Binary Trees :→

→ The storage representation of Binary trees in two ways

(i) Sequential Representation.

(ii) Linked list Representation.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Sequential Representation :→

→ The sequential representation is nothing but arrays.

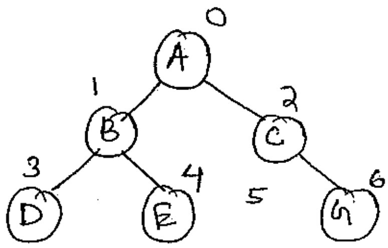
→ An array can be used to store the nodes of a binary tree.

→ The nodes can be accessed sequentially.

→ Suppose a binary tree "T" of depth "d" the atmost $(2^d - 1)$ nodes can be there in T.

→ so the array size to represent a binary tree of ~~deft~~ depth 3 is $2^3 - 1 = 7$ so A[7]

→ To perform the operations, we have to identify root, left child and Right child.



(i) The root of a node having index 'n' can be obtain by $(n-1)/2$.

for example :- To find the root of B where array index $n = 3$, then the root node index can be obtained as

$$(3-1)/2 = 2/2 = 1.$$

i.e A[1] is the root of D which is B.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

20/12/2016

(ii) The left child of a node having an index 'n' can be obtained by $(2n+1)$.

Example :- To find the left child of C where array index $n = 2$, this can be obtained by $(2n+1) = (2\times2+1) = 5$ i.e $A[5]$ is the left child of C which is null. so as per the fig. there is no left child of C.

(iii) The right child of a node having array index 'n' can be obtained by the formula $(2n+2)$.

Example :- To find the right child of the B, where array index $n=1$, this can be obtained by $2n+2 = 2\times1+2 = 4$, i.e $A[4]$ is the right child of B which is E.

(iv) If the left child is a array index n, i.e right child is $(n+1)$. similarly if a right child of index n then its left child is $(n-1)$

The above tree is not good representations by using ~~arar~~ arrays it is better to represent in complete binary trees.

Linked List Representation :-

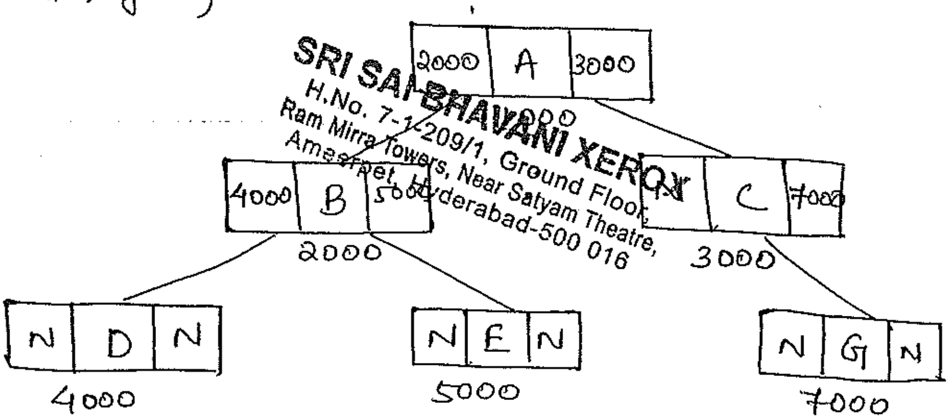
→ The most populer way and practical way of representating a binary tree using Linked list concept.

→ To implement the trees a node will contain minimum 3 fields

(i) Information field.
(ii) Previous field / left child
(iii) Next field / Right child.

prev	info	next

```
struct node tree
{
   struct tree *left;
   int info;
   struct tree *right;
};
```



Operations on Binary trees :-

1. Creating a Binary Tree
2. Traversing a Binary tree
3. Inserting a node
4. Deleting a node
5. Searching for a Node.
6. copying the mirror image of a tree
7. Determine the total number of nodes
8. Determine the total number of leaf nodes.
9. Determine the total number of non-leaf nodes.
10. finding smallest element in a Node.
11. finding Largest element.
12. Find the height of the tree.
13. finding the father / Left child / Right child / Brother of an arbitary node.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Traversing a Binary Tree :→

→ Tree traversal is one of the most common operation, perform the tree datastructures, it is a way in which each node in the tree is visited exactly only once in a systematic manner.

→ They are the standard base to represent tree traversals

Like :- → Pre-order
→ Inorder
→ postorder.

→ we can also have Level order.

→ This traversal techniques can be implemented in two ways

(i) Iteration
(ii) Recursion

With Recursion of preorder :→

→ To traverse a non-empty binary tree in pre-order following steps are to be processed.

1. Visit the root node.
2. Traverse the left sub tree in preorder
3. Traverse the right sub tree in preorder.

→ That is in preorder traversal, the root node is visited (or processed) first, before traveling through left and right sub tree recursively.

With iteration :→

Algorithm for preorder Traversing :→

Step 1 : set TOP ← 1
Stack [1] ← NULL
PTR ← Root

Step 2 : Repeat 3-5 while PTR != NULL

Step 3 : Apply process to info [PTR]

Step 4 : if Right [PTR] != NULL then
TOP ← TOP + 1
Stack [TOP] ← Right [PTR]

Step 5: if Left[PTR] != NULL then

PTR ← Left [PTR]

else

PTR ← stack [Top]

Top ← Top − 1

Inorder Traversal Recursively :→

→ The inorder traversal of a non-empty binary tree is defined as follows :-

1. Traverse the left sub tree in order
2. Visit the root node
3. Traverse the right sub tree in order

→ Inorder traversal, the left sub tree is traversed recursively, before visiting the root.

→ After visiting the root the right sub tree is traversed recursively in order fashion.

Algorithm :-

← (assigning)

Step 1: set TOP ← 1

Stack [1] ← NULL

PTR ← Root

Step 2: Repeat while PTR != NULL

Top ← Top + 1

stack [TOP] ← PTR

PTR ← Left [PTR]

Step 3: PTR ← stack [Top]

Top ← Top − 1

Step 4: Repeat steps 5-7 while PTR != NULL

Step 5: Apply process to Info [PTR]

Step 6: if Right [PTR] != NULL then

PTR ← Right [PTR]

Go to step 2

Step 7: PTR ← stack [Top]

TOP ← TOP − 1

Post order Traversal :→

The post order traversal is a non-empty binary tree can be defined as :

1. Traverse the left sub tree in post order
2. Traverse the right sub tree in post order.
3. Visit the root node.

In post order traversal, the left and right sub tree(s) are recursively processed before visiting the root.

Algorithm :→

Step 1 : Set TOP ← 1
  stack [1] ← NULL
  PTR ← Root

Step 2 : Repeat 2-5 while PTR != NULL

Step 3 : Top ← Top + 1
  stack [TOP] ← PTR

Step 4 : if Right [PTR] != NULL
  TOP ← TOP + 1
  stack [TOP] ← Right [PTR]

Step 5 : PTR ← Left [PTR]

Step 6 : PTR ← stack [TOP]
  Top ← Top − 1

Step 7 : Repeat while PTR > 0
  Apply process to Info [PTR]
  PTR ← stack [TOP]
  TOP ← TOP − 1

Step 8 : if PTR < 0
  PTR ← ~~PTR~~ − PTR
  Go to step 2.

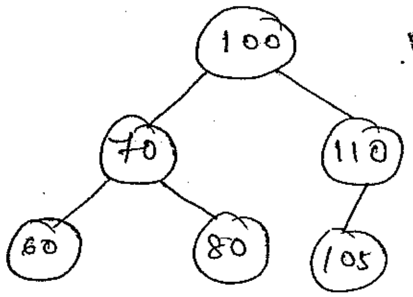
→ The level order of traversal, visiting every node from level 1 to level n.

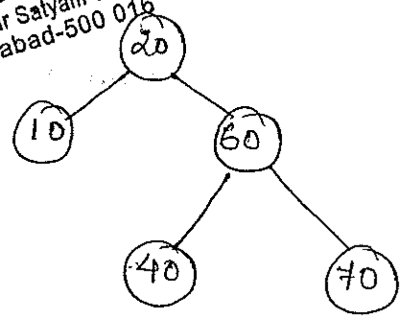
→ Before implementing the traversal technique try to implement any binary trees.

Binary Search Tree :→ (BST)

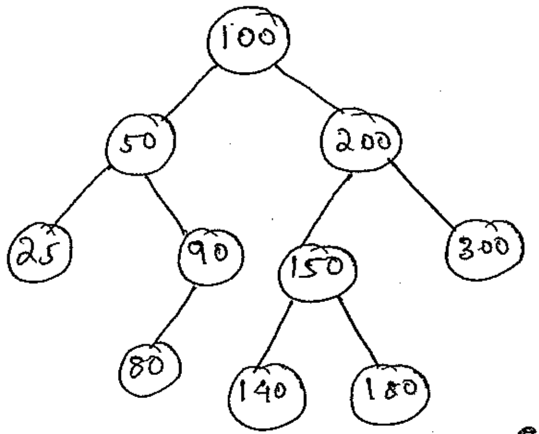
A binary search tree is a binary tree in which for each node in the tree, elements in the left sub tree are less than root and elements in the right sub tree are greater than root.

Example :-





Ex: 100, 50, 200, 90, 80, 25, 300, 150, 180, 140



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

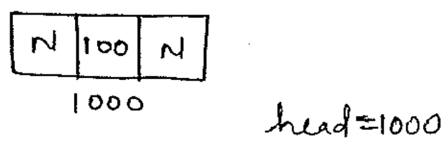
21/12/2016

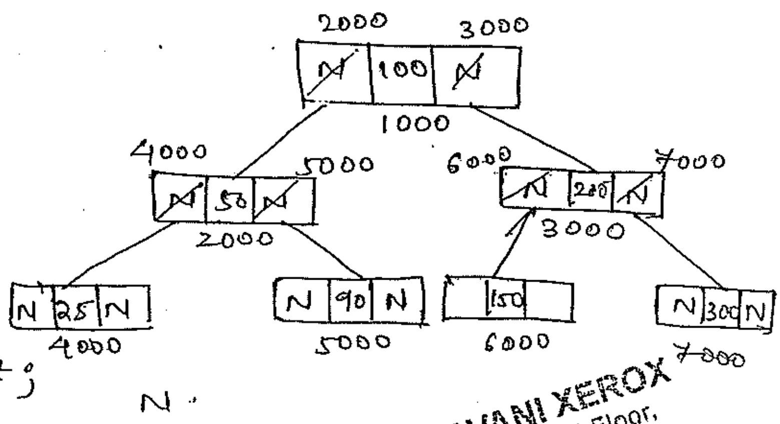
Q) Write a program on BST with implementing tree traversal techniques with iteration concept.

```
Struct tree
 {
  struct tree *left;
  int info;
  struct tree *right;
 };
Struct tree * root = NULL;
Struct tree * create();
Void display (struct tree *);
void display (struct tree *);
void preorder (struct tree *);
void inorder (struct tree *);
void postorder (struct tree *);
main()
{
  int ch;
  do
   {
    a:
     clrscr();
    printf ("\n CHOICE ACTION \n\n");
    printf ("1. creation of Tree \n\n");
    printf (" 2. Display of Tree \n");
    printf ("3. Exit \n");
    switch (ch)
     {
      case 1 : root = create();
               ch=0;
               goto a;
      case 2 : display (root);
               ch=0;
               getch();
      case 3 : exit (0);
     }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
    printf ("\n Enter your choice : ");
    scanf ("%d",&ch);
   }
  while (ch!=3);
 }
struct tree * create()
 {
   struct tree * head,
   int n,x,
   clrscr();
 printf ("\n Enter how many nodes do you want to create :");
 scanf("%d",&n);
 printf ("\n Enter the data into 1 Node :");
 scanf ("%d",&x);
 head=(struct tree*)malloc(sizeof(struct tree*));
 head->info=x;
 head->left = NULL;
 head->right = NULL;
 for(i=1; i<n; i++)
 {
  printf ("\n Enter Data into %d Node :",i+1);
  scanf("%d",&x);
  first=head;
  while (first != NULL)
   {
    prev=first;
    if(first->info>x)
     {
      first=first->left;
     }
    else
     {
      first = first->right;
     }
   } //while
```





```
temp=(struct tree *)malloc(sizeof(struct tree));
  temp->info=x;
  temp->left=NULL;
  temp->right=NULL;
 if (prev->info >x)
   {
    prev->left=temp;
   }
 else
   {
    prev->right=temp;
   }
 } //for
return head;
}
 void display(struct tree *ptr)
  {
   int t;
   clrscr();
   printf("\n In which method the tree has to traverse \n\n");
   printf("1 PREORDER \n\n");
   printf("2 INORDER \n");
   printf("3 POSTORDER \n");
   printf("Enter your choice :");
   scanf("%d",&t);
   switch(t)
    {
     case 1: preorder(ptr);
             break;
     case 2: inorder(ptr);
             break;
     case 3: postorder(ptr);
             break;
     default: printf("Error...");
  } }
```

→ Pre order NLR
100, 50, 25, 90, 200, 150, 300
→ post order LRN
25, 90, 50, 150, 300, 200, 100
→ Inorder LNR
25, 50, 90, 100, 150, 200, 300

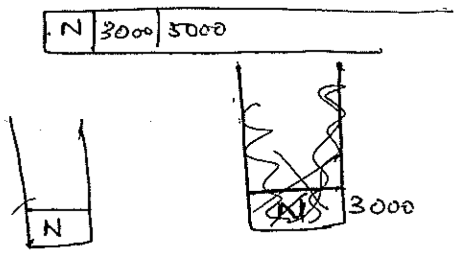
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

22/12/2016

```
Void preorder (struct tree * ptr)
{
 int top = 0;
 struct tree * first = ptr, *stack[20];
 stack[top] = NULL;
 while ( first != NULL)
 {
  printf ("%3d", first->info);
  if (first->right != NULL)
  {
   top++;
   stack[top] = first->right;
  }
  if ( first -> left != NULL)
  {
   first = first -> left;
  }
  else
  {
   first = stack[top];
   top--;
  }
 }
}
```

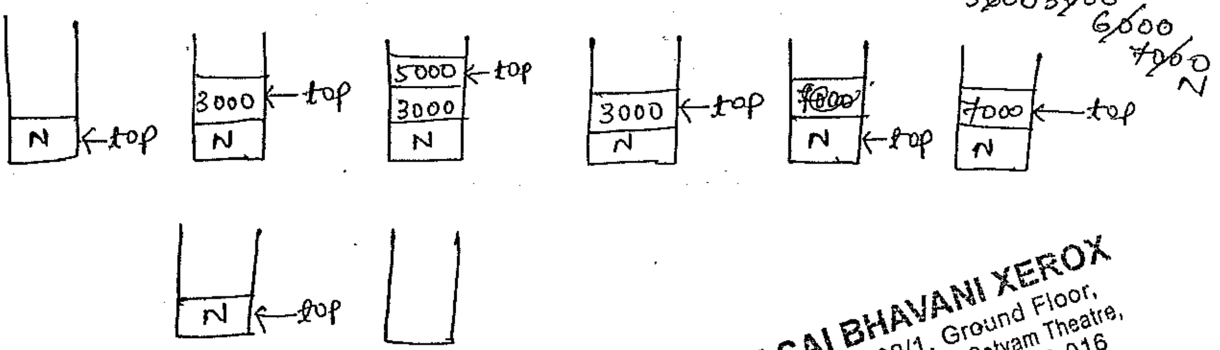
top = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ 0
S[0] = ~~NULL~~; 3000
ptr = 1000
first = ~~1000~~ ~~2000~~ ~~5000~~ ~~4000~~ ~~3000~~ ~~6000~~ 7000
S[1] = 3000
S[2] = 500
S[1] = 7000

..., 50, 25, 90, 200, 150,



top = ~~0~~ 1
S[0] = NULL. ptr = 1000
first = ~~1000~~ ~~2000~~ ~~4000~~ ~~5000~~ ~~3000~~ ~~6000~~ ~~7000~~ N

void Inorder (struct



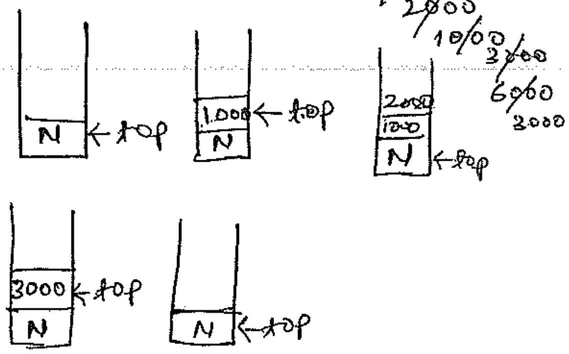
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
void inorder ( struct tree *ptr)
{
  struct tree * stack[20];
  int top= 0;
  Stack[top] = NULL;

abc:
  while (ptr! = NULL)
   {
    top = top+1;
    Stack [top] = ptr;
    ptr = ptr -> left;
   }
  ptr = Stack[top];
  top--;
 while (ptr! = NULL)
  {
   printf ("%3d", ptr -> info);
   if (ptr -> right! = NULL)
    {
     ptr = ptr -> right;
     goto abc;
    }
   ptr = stack[top];
   top--;
  }
}
```

top = ~~0~~ ~~1~~ ~~0~~ 1

ptr = ~~1000~~, ~~2000~~, ~~1000~~, ~~3000~~, ~~6000~~, 3000



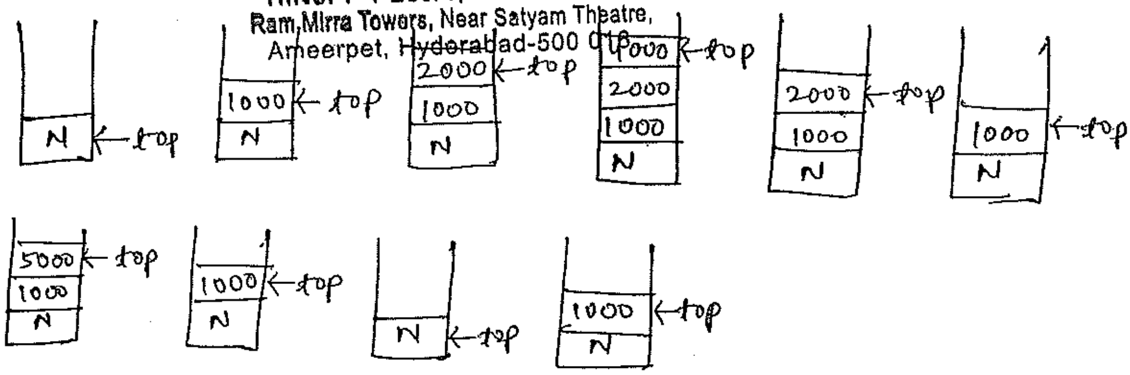
~~100~~, 200

top = ~~0~~ 1

S[0] = 0

ptr = ~~1000~~, ~~2000~~, ~~4000~~, ~~N~~, ~~4000~~, ~~2000~~, ~~5000~~, ~~N~~, ~~5000~~, 1000

25 50 90



```
Void postorder(struct tree * ptr)
{
    int top = 0;
    struct tree * stack[20];
    stack[top] = NULL;
abc:
  while (ptr != NULL)
  {
     top++;
     stack[top] = ptr;
     if(ptr->right != NULL)
     {
        top++;
        stack[top] = -(ptr->right);
     }
     ptr = ptr->left;
  }
  ptr = stack[top]
  top--;
  while(ptr > 0)
  {
     printf("%d", ptr->info);
     ptr = stack[top];
     top--;
  }
  if(ptr < 0)
  {
     ptr = -ptr;
     goto abc;
  }
  }
}
```

```
Void postorder(struct tree * root)
{
long int stack[20], pointer;
struct tree * ptr;
int top = 0;
ptr = root;
stack[top] = 0;
while (ptr != NULL)
  {
   top++;
   stack[top] = (int) ptr;
   if (ptr->right != NULL)
     {
       top++;
       stack[top] = -(int)(ptr->right);
     }
   ptr = ptr->left;
   top--;
   while (pointer > 0)
   {
   ptr = (struct tree *)pointer;
   printf("%d\t", ptr->info);
   pointer = stack[top];
   top--;
   }
   if (pointer < 0)
    {
     pointer = - pointer;
     ptr = (struct tree *)pointer;
     goto abc;
    }
  }
}
```

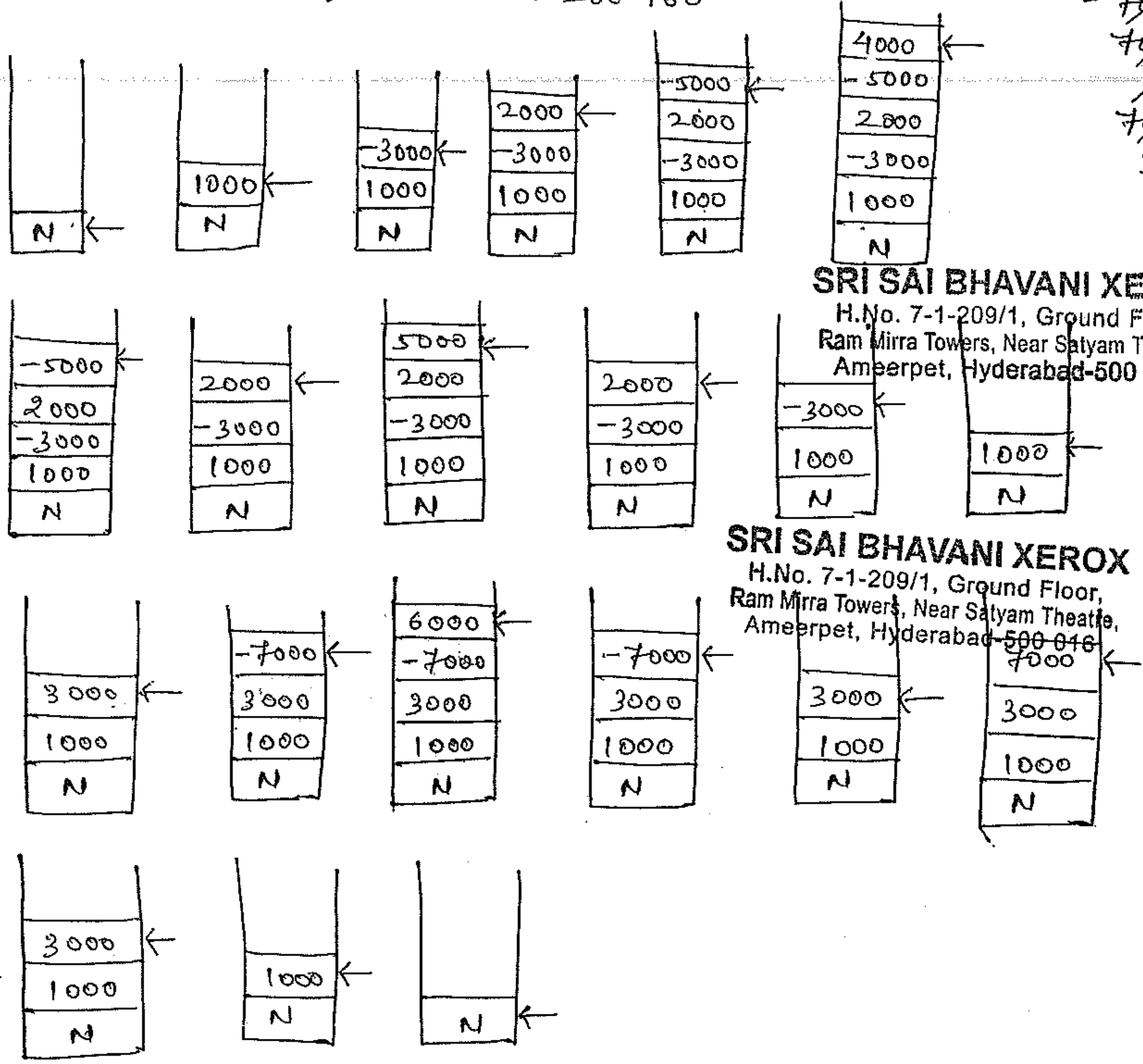

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ptr = ~~1000~~ ~~2000~~ ~~4000~~ ~~N~~ ~~4000~~ – ~~5000~~ ~~5000~~ ~~N~~ ~~5000~~ ~~2000~~ – ~~3000~~ ~~3000~~ ~~6000~~ ~~N~~

top = 0

25 90 50 150 300 200 100

~~6000~~
– ~~7000~~
~~7000~~
~~N~~
~~7000~~
~~3000~~
~~1000~~
N



23/12/2016

Q) Implement the BST of creation or insertion, find, Delete operations.

```
#include <stdio.h>
# include <malloc.h>
struct node
 {
  int info;
  struct node *lchild;
  struct node *rchield;
 } *root;
main()
{
 int choice, num
 root = NULL;
while (1)
 {
 printf("\n");
 printf(" 1. Insert \n");
 printf(" 2. Delete \n");
 printf(" 3. Inorder Traversal \n");
 printf(" 4. Display \n");
 printf (" 5. Quit \n");
 printf ("Enter your choice :");
 scanf (" %d", &choice);
 switch (choice)
  {
  case 1 :  printf ("Enter the number to be inserted : ");
            scanf ("%d", &num);
            insert (num);
            break;
  case 2 : printf ("Enter the number to be deleted");
           scanf (" %d ", &num);
           del (num);
           break;
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
case 3: inorder(root);
        break;
case 4: display(root,1);
        break;
case 5: exit(0);
    default:
            printf("wrong choice\n");
  }
 }
}
find(int item, struct node **par, struct node *loc)
 {
  struct node *ptr, *ptrsave;
     if(root==NULL) // tree empty
      {
       *loc = NULL;
       *par = NULL;
        return;
      }
     if(item== root->info) // item is at root.
      {
       *loc =root;
       * par=NULL;
         return;
      }
 // initialize ptr and ptrsave
     if (item < root->info)
        ptr =root -> lchild;
        else
        ptr= root-> rchild;

        ptrsave =root;
     while(ptr!=NULL)
      {
        if(item==ptr->info)
          {
           *loc =ptr;
           *par=ptrsave;
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

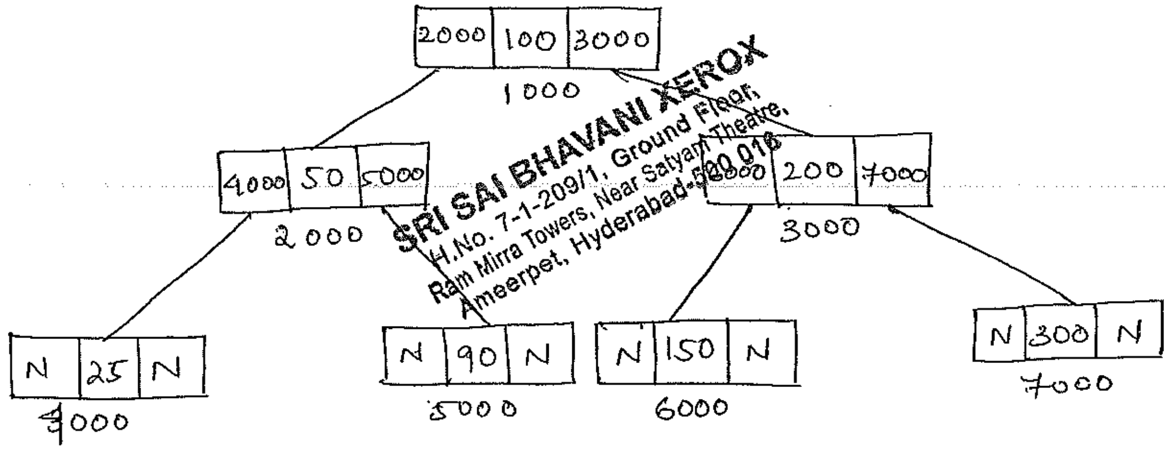
```
        return;
      }
   ptrsave = ptr;
    if ( item < ptr->info)
      ptr=ptr->lchild;
      else
       ptr=ptr->rchild;
 } //while

   *loc = NULL; // item not found
   *par = ptrsave;
```

24/12/2016

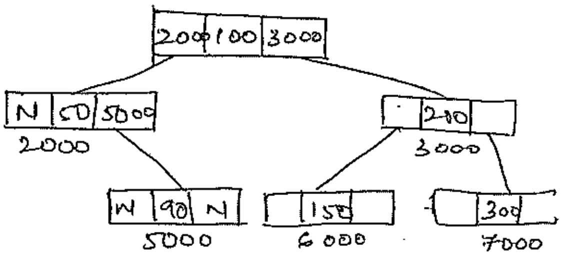
```
insert(int item)
{
 insert node *tmp , *parent , *location ;
find ( item , &parent , &location);
  if (location != NULL)
   {
   printf ("Item already existed ");
    return;
   }
 tmp= (struct node *)malloc (sizeof (struct node));
  tmp->info = item ;
  tmp->lchild = NULL;
  tmp->rchild = NULL;
  if ( parent == NULL)
     root = tmp;
else
   if (item < parent ->info)
        parent ->lchild =tmp;
      else
        parent ->rchild =tmp;
  }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016




```
del (int item)
{
 struct node *parent, *location;
 if (root == NULL)
  {
   printf("Tree empty");
   return;
  }
find (item, &parent, &location);
 if (location == NULL)
 {
 printf("Item not present in tree");
 return;
 }
if (location->lchild == NULL && location->rchild == NULL)
        case_a(parent, location);
if (location->lchild != NULL && location->rchild == NULL)
        case_b(parent, location);
if (location->lchild == NULL & location->rchild != NULL)
        case_b(parent, location);
if (location->lchild != NULL && location->rchild != NULL)
        case_c(parent, location);
 free(location);
 }
```

```
case_a(struct node *ptr , struct node *loc) //2000  4000
{ if (par==NULL) // item to be deleted is root node
     root=NULL;
  else
     if (loc==par->lchild)
        par->lchild=NULL;
     else
        par->rchild=NULL;
} // case_a
```




```
case_b (struct node *par , struct node *loc) //1000  2000
{
  struct node *child;
  // initialize child
 if (loc->lchild!=NULL)
   child=loc->lchild;
 else
   child = loc->rchild; //5000
  if (par==NULL) // item to be deleted is root node.
     root=child;
  else
    if (loc==par->lchild)
      par->lchild=child;
  else
      par->rchild=child;

 } // case_b
case_c (struct node *par, struct node *loc) //1000  3000
{
 struct node *ptr , *ptrsave , *suc , *parsuc;
 // find inorder successor and its parent
    ptrsave=loc; //3000
    ptr=loc->lchild ; //6000
   while (ptr->lchild!=NULL)
     {
       ptrsave=ptr;
       ptr=ptr->lchild;
     }
```

```
suc = ptr;  //6000
parsuc = ptrsave ; //3000
if(suc->lchild == NULL && suc->rchild == NULL)
    case_a (parsuc, suc); //3000 6000
  else
    case_b (parsuc, suc);
if (par == NULL) // if item to be deleted is root node.
  root = suc;
else
  if ( loc == par->lchild)
   par->lchild = suc;
else
   par->rchild = suc;
 suc->lchild = loc->lchild;
 suc->rchild = loc->rchild;
} //case_c
 inorder (struct node *ptr)
  {
   if (root == NULL)
    {
     printf ("Tree is empty");
     return;
    }
   if (ptr != NULL)
    {
     inorder (ptr->lchild);
     printf (" %d ", ptr->info);
     inorder (ptr->rchild);
    }
  } //inorder.
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
display (struct node * ptr , int level)
{
  int i;
  if (ptr != NULL)
  {
    display (ptr->rchild , level+1);
    printf ("\n");
    for( i=0; i<level; i++)
      printf ("    ");
    printf ("%d", ptr->info);
    display (ptr->lchild , level+1);
  } //if
} // display.
```


31/12/2016

Expression Trees :-

→ An expression tree for an arithmatic relational or logical expressions can store in an expression trees.

→ The expression trees also from the family of binary trees

→ when an expression tree is forming we need to follow the rules, the paranthesis in the expressions donot appear.

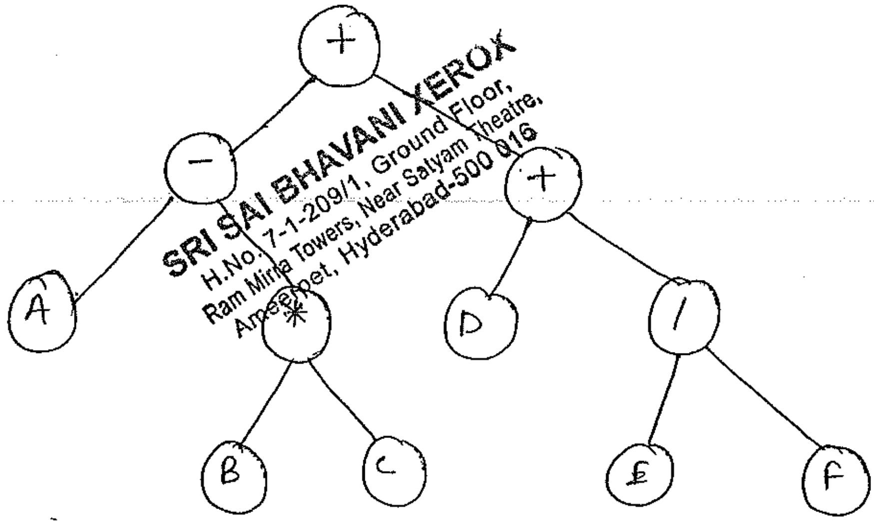
→ The leaves are ~~of~~ the variables ~~f~~or the constant in the expression.

→ The non-leaf ~~leaves~~ nodes are the operators in the expression tree.

→ A node for binary operators has two non empty sub-trees.

→ A node for unary operator has one non-empty sub-tree.

→ In a ~~the~~ given expression, the operators with the least priority will acts a root node (ascending to descending).

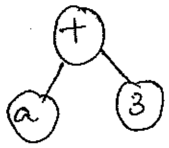


Infix Notation : ((A-B*C) + (D+E/F))

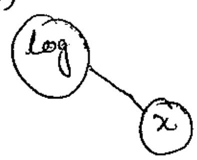
→ The operators, constants and variables are arranged in such a way that an inorder traversal produces the original expression without paranthesis.

→ The least priority of operator will acts as a root node and followed by other operator.

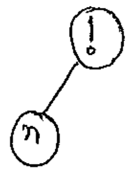
Ex: (a+3)



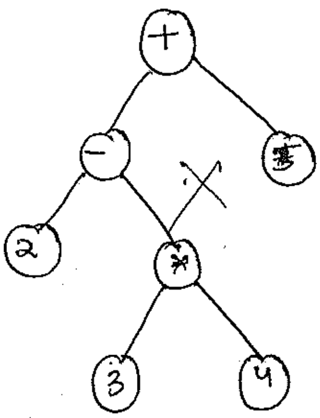
log(x)

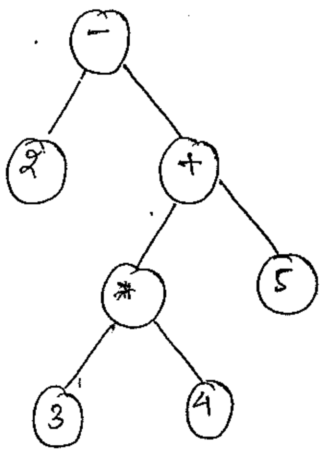


n!



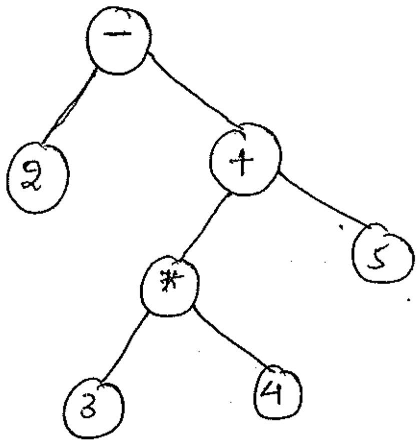
2 - 3 * 4 + 5



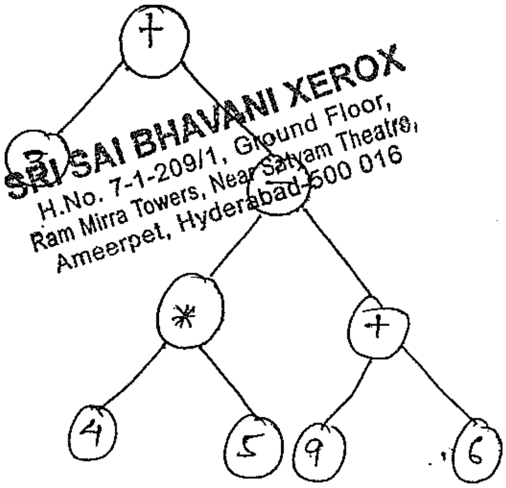


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

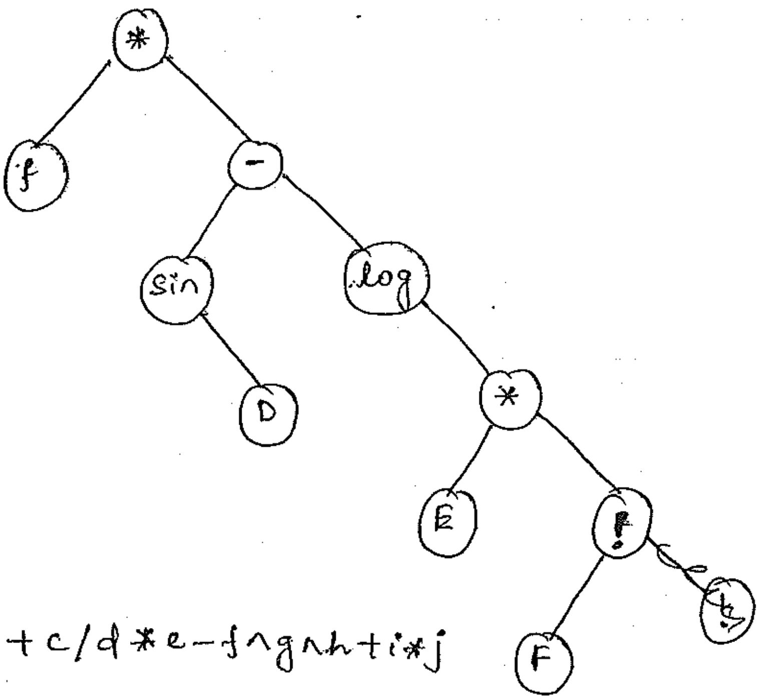
$2-(3*4+5)$



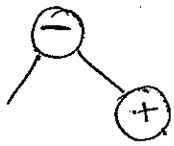
$3+(4*5-(9+6))$



$f(A,B,C)*(\sin(D)-\log(E*F!))$



$a=-b+c/d*e-f\wedge g\wedge h+i*j$



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

why expression trees required?

→ for evaluations of expression, generating Correct compiler code to actually compute the expression value and execution time.

→ Basically all the compilers will convert into expression trees at the time of evaluation.

→ perfoming symbolic mathematical operations on the expressions

2/01/2017

1. Implementation of an expression tree to perform tree traversals.

AIM:-

To implement an expression tree and to perform pre-order in-order and post-order traversals.

Algorithm :-

Step1 :- Start the process

step2 :- Initialize and declare variables

step3 :- Enter the postfix expression that can be stored in the stack.

Step4 :- In pop operation, check the top of the stack is empty; otherwise stack[top] = Node, and decrement the top value.

step5 :- In push operation, check the size of the stack, stack is not full, top is incremented and stack[top]=Node

step6 :- Allocate the memory for new character and assigned left and right pointer is NULL.

step7 :- If character value is +, *, /, -, pop the right and left pointer to temp and push into stack

step8 :- The preorder function is traverse the root node and left, right node of the tree.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```c
#include<stdio.h>
# include<conio.h>
# include<alloc.h>
# include<ctype.h>
#define size 20
typedef struct node
{
 char data;
 struct node *left;
 struct node *right;
 } btree;
 btree *start[size];
   int top;
 void main()
  {
   btree *root;
   char exp[80];
   btree *create(char exp[80]);
   void preorder(btree *root);
   clrscr();
   printf("Enter the postfix expression ");
   scanf("%s", exp);
   top=-1;
   root=create(exp);
   printf("\n The tree is created ");
   printf("\n The preorder traversal of tree \n");
   preorder(root);
   getch();
   }
```


SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016