Expression Tree

Expression tree is a binary tree wherein a give infix expression can be represented as a binary tree with operands as leaf nodes and operators as internal nodes.

Algorithm to construct an exp tree.

Step 2 → Scan the given expression from left to right.

Step 1 → Initialize 2 stacks namely a tree stack and operator stack.

Step 3 If the scanned symbol is

(a) an operand - construct a node for the operand & push the node onto the tree stack.

(b) an operator - if operator stack is empty or the precedence of the operator on the top of the operator stack is less than the precedence of the scanned operator, construct a node for the operator & push it onto the operator stack.

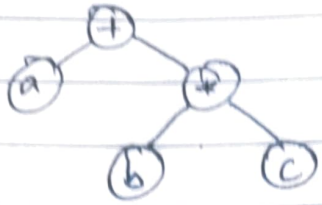
else pop 2 nodes from the tree stack & attach them as the right & the left child & push the operator node onto the tree stack and push the scanned operator onto the operator stack.

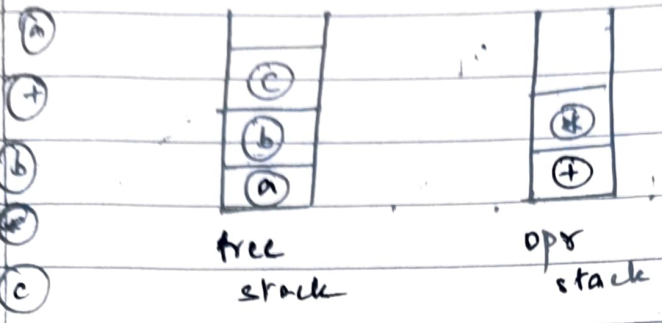
(pop an operator node from the operator stack).

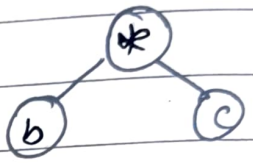
Step 4 Until the operator stack becomes empty, pop an operator node from the operator stack and 2 nodes from the tree stack, attaching them as the right and the left child and push the operator node onto the tree stack.

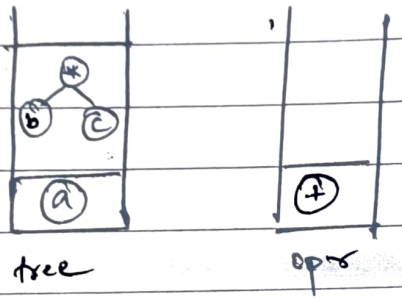
Step 5 Return tree stack to get your exp. tree.

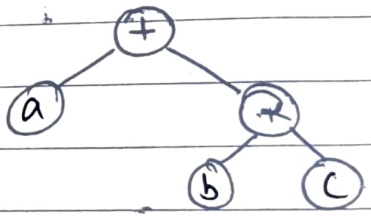
a + b * c

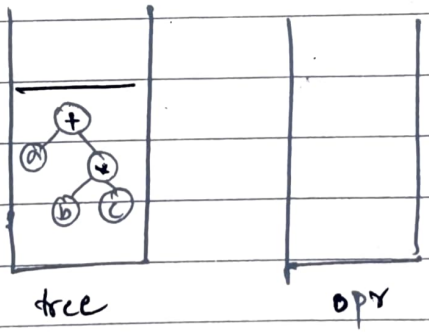




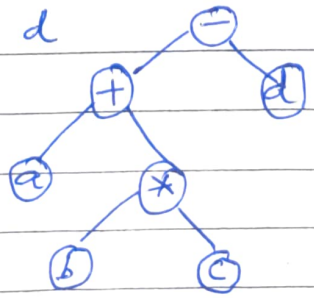








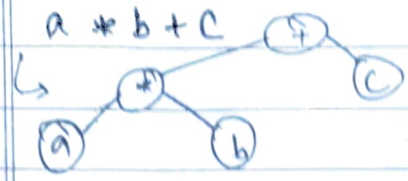
a + b * c - d

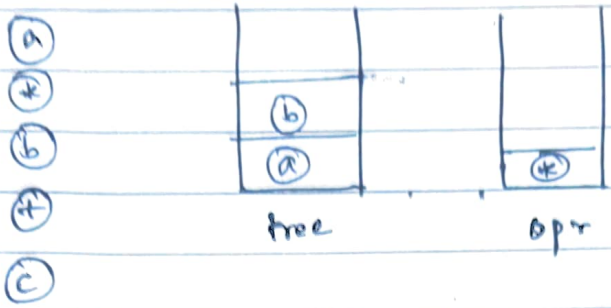


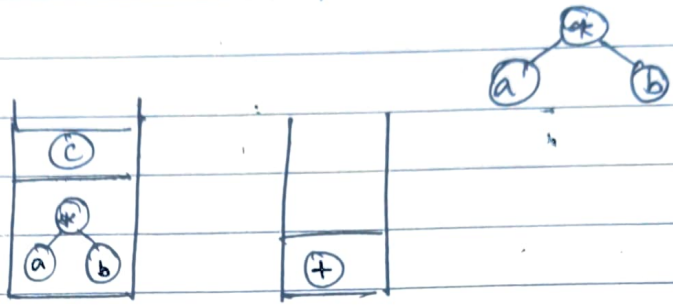
pre → - + a * b c d

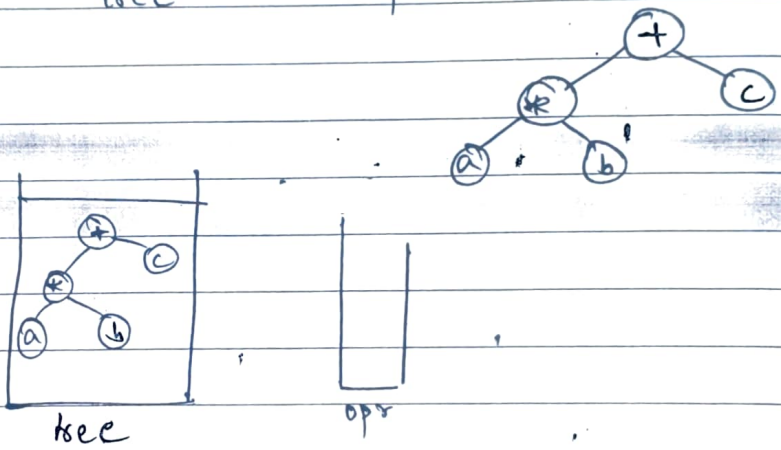
post → a b c * + d -

Q a * b + c









Q LAB-8

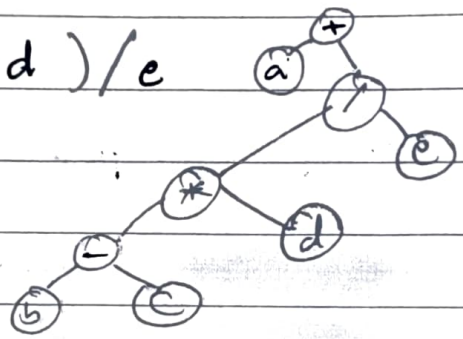
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
struct node
{ char data;
  struct node *left;
  struct node *right;
};
typedef struct node
```

```
NODE create_node (char item)
{
  NODE temp;
  temp = (NODE) malloc (sizeof (struct node));
  temp -> data = item;
  temp -> lft = NULL;
  temp -> right = NULL;
  return temp;
}

int precede (char c)
{
  switch (c)
  { case '$': return 5;
    case '*':
    case '/': return 3;
    case '+':
    case '-': return 1;
  }
}

NODE createexptree (char infix[15])
{  char symbol
   NODE treestack[20], opstack[20], temp, t, l, r;
   int top1 = -1, top2 = -1, i,
   for (i = 0; infix[i] != '0'; i++)
     {
       symbol = infix[i];
       if(isalnum(symbol))
         {
           temp= create_node(symbol);
           treestack[++top1] = temp;
         }
```

```
else
 {
    temp = creat_node (symbol);
    if (top2 == -1) || preced(symbol) > preced(opstack[top2]->data))
        opstack [++top2] = temp;
    else while (preced(symbol) <= preced(opstack [top2]->data))
    { t = opstack [top2--];
      r = treestack [top1--];
      l = treestack [top1--];
      t->right = r;
      t->left = l;
      treestack [++top1] = t; }
    opstack[++top2] = temp;
 }
}
while ( top2 != -1)
{
    t = opstack [top2--];
    r = treestack [top1--];
    l = treestack [top1--];
    t->right = r;
    t->left = l;
    treestack [++top1] = t;
 }
 return treestack [top1];
}
// write preorder, inorder, postorder
// change %d to %c

int main()
{
   NODE root = NULL;
   char infix[20];
```

```
printf("\n Read the expression \n");
scanf("%s", infix);
root = createexptree(infix);
printf("\n Preorder traversal is \n");
preorder(root);
printf("\n Inorder traversal is \n");
inorder(root);
printf("\n Postorder traversal is \n");
postorder(root);
}
```

Q a + ((b-c) *d)/e

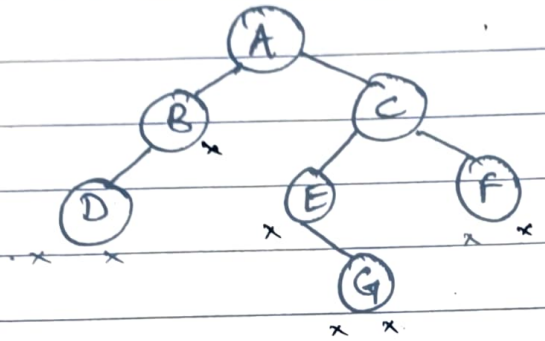


Q Write a short note on threaded binary tree.

Threaded Binary Trees

Drawback of bin tree

- 50% of the address are stored NULL.
- time consumed is more for traversal operation
- To find the successor / predecessor node it takes more time.



To overcome this, we go for threaded bin trees. (based on traversal)

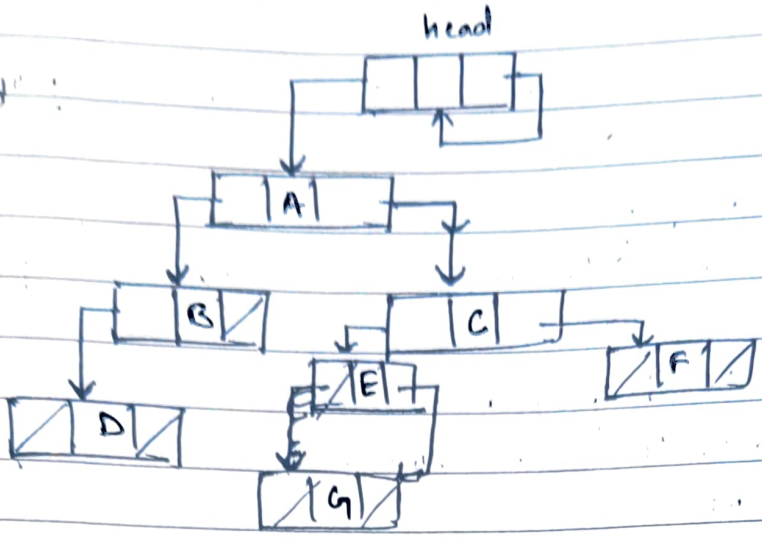
consists of head node which contains addr of root node.

1. In-threaded bin tree
2. Post - " "
3. Pre- " "

2 types

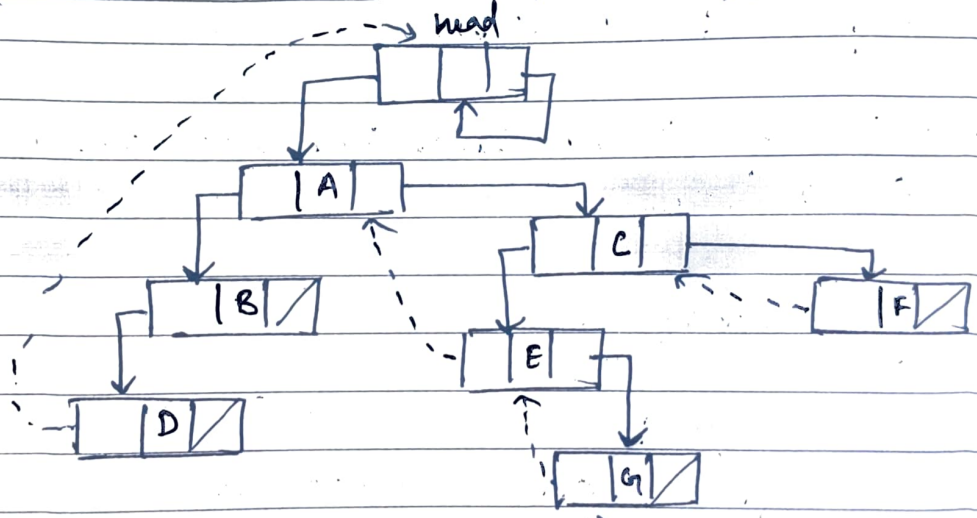
left → predecessor

right → successor

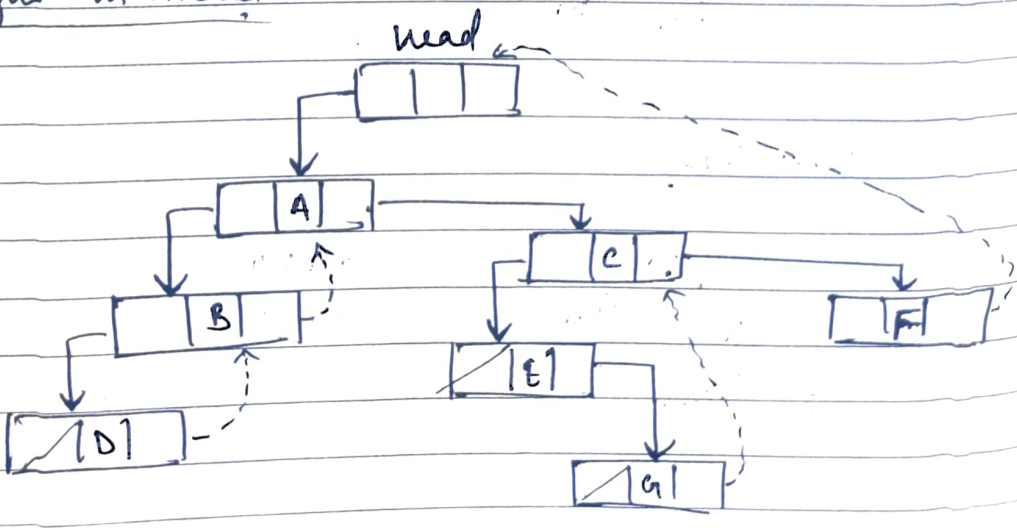


Inorder : D B A E G C F

Left in-threaded bin tree (predecessor)



Right in-threaded bin tree (successor)



According to inorder

Heap

Heap is a binary tree with 2 additional properties :

(1) Structural property - → except last level all levels 2^i node.
The tree should be almost complete binary tree.

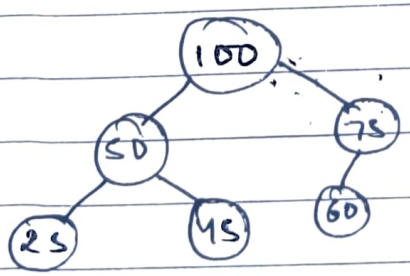
(2) Parent dominant property -
Parent node should be more dominant when compared to its children.

There are 2 types of heaps :

(1) Max heap - Parent should have higher value when compared to its children.

(2) Min heap - Parent should have lesser value compared to its children.

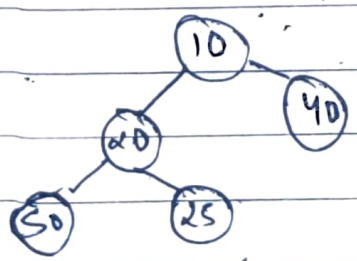
eg: Max heap



50 & 75 < 100

25 & 45 < 50

Min heap



10 & 40 > 10

50 & 25 > 20

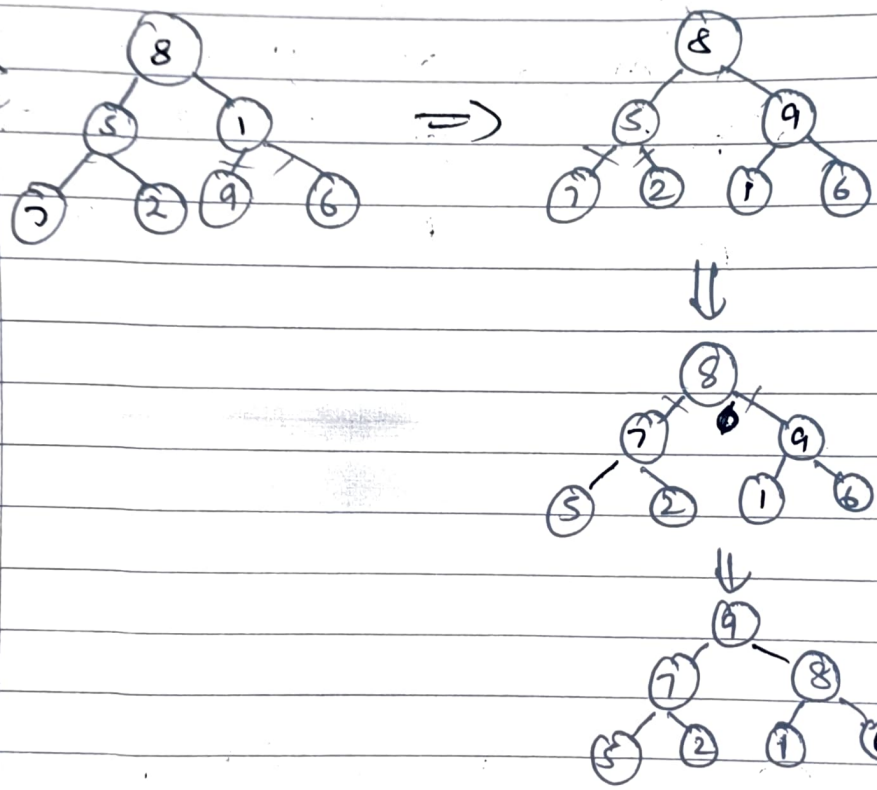
There are 2 methods to construct a heap.

1. Bottom-up method
2. Top down method

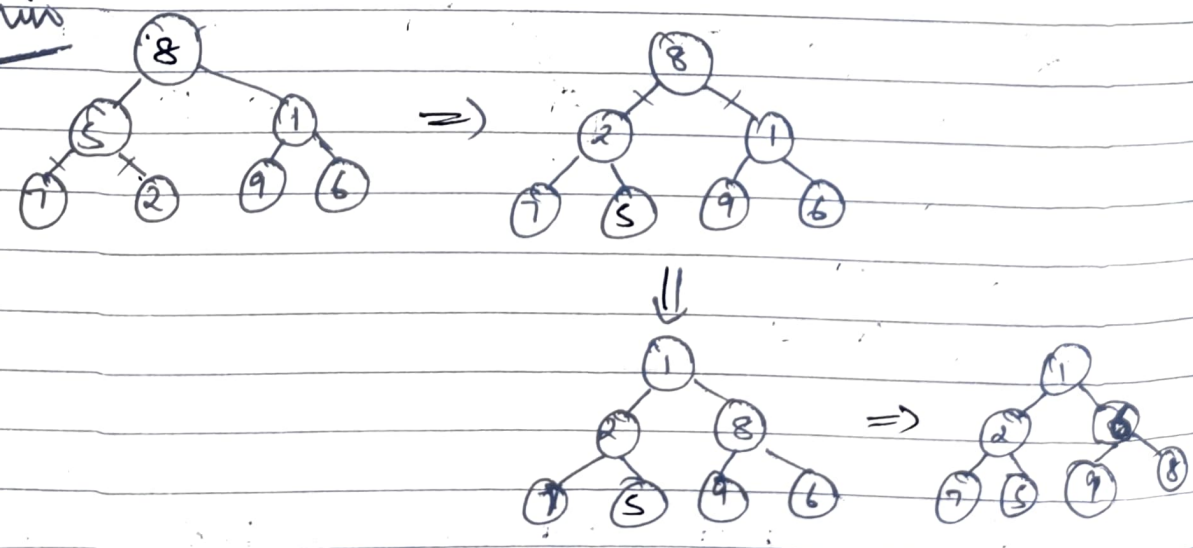
Bottom up.

Construct a max heap for the following elements:
8 5 1 7 2 9 6

Max



Min



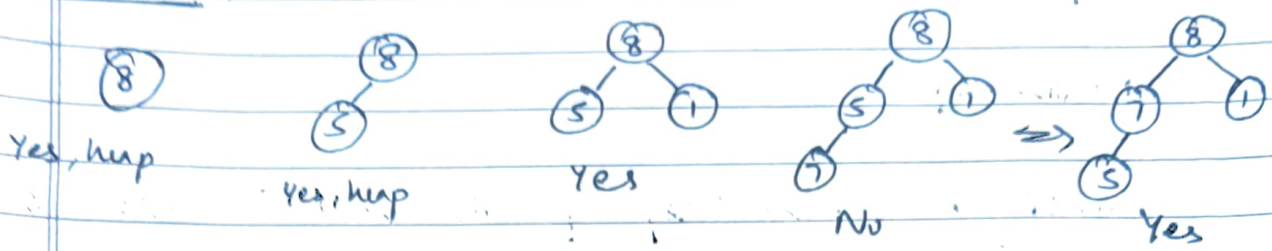
* Not a compulsion that you'll get same heap using both methods.

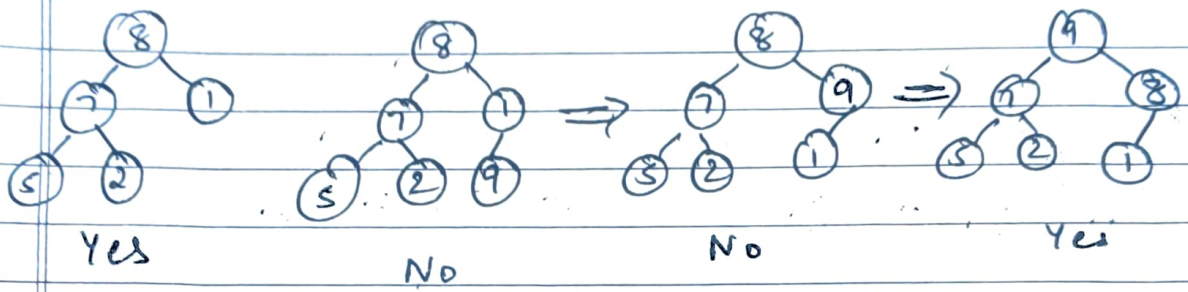
classmate
Date
Page

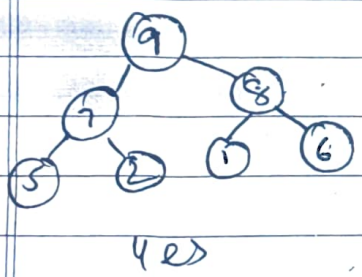
Top down.

8 5 1 7 2 9 6

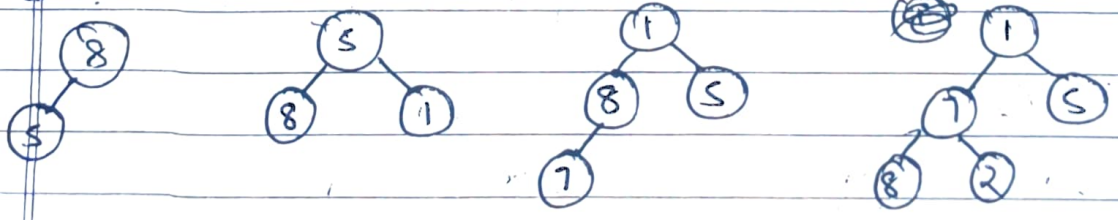
Max

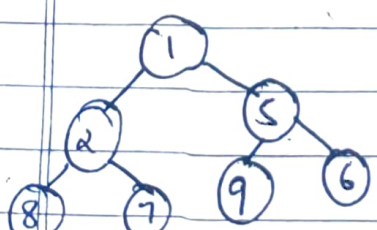






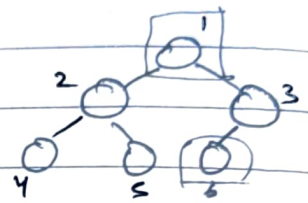
Min





Program to implement priority queue using heap.

```
#include <stdio.h>
#include <stdlib.h>
int n;
int extractmax(int a[10])
{ int max
  if (n == 0)
  {
     printf("Heap is empty");
     return -1;
  }
  else
  { max = a[1];
     a[1] = a[n];
     n = n-1;
     heapify(a,1);
     return max;
  }
}
void buildheap(int a[10])
{ int i
  for (i = n/2; i >= 1; i--)
     heapify(a,i);
}
void heapify(int a[10], int i)
{ int left, right, largest;
  left = 2*i;
  right = 2*i+1;
  if (left <= n && a[left] > a[i])
     largest = left;
  else
     largest = i;
  if(right <= n && a[right] > a[largest])
     largest = right;
  if (largest != i)
  {
     swap(&a[i], &a[largest]);
     heapify(a,largest);
```



position

```
int main()
{  int a[10], i, ch, del
   printf("\n Read no. of elements
   printf("1. To creati heap \n 2. Delete \n 3. Exit \n");
   printf("Read choice \n");
   scanf("%d", &ch);
   switch(ch)
   {  case 1: printf("Read no. of elements \n");
              scanf("%d", &n);
              printf("Read elements \n");
              for(i=1; i<=n; i++)
                   scanf("%d", &a[i]);
              buildheap(a);
              printf("Elements of the constructing heap \n");
              for(i=1; i<=n; i++)
                   printf("%d \t", a[i]);
              break;

      case 2: del = extractmax(a);
              if(del != -1)
                   printf("Element deleted is %d \n", del
              printf("\n Elements after deletion \n");
              for(i=1; i<=n; i++)
                   printf("%d \t", a[i]);
              break;
      default: exit(0);
   }
}

void swap(int *a, int *b)
{  int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
```