# Data Structure and Algorithm Lab File

| | | |
|---|---|---|
| Name | : | Yash Jain |
| Branch | : | Computer Science |
| Roll No. | : | 11911029 |

# INDEX

## Create a singly linked list of integers

```
#include <stdio.h>
#include <stdlib.h>
//Declaring a  singly list node structure
struct node
{
int data;
struct node *link;
}*first;
//traversing through linked list and displaying its elements
void display(struct node *p)
{
while(p)
{
printf ("%d\t",p->data);
p=p->link;
}
}
//creating SLL using array
int main()
{
int A[5];
struct node *p,*t;
p=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[0]);
p->data=A[0];
p->link=NULL;
first=p;
for(int i=1;i<5;i++)
{
t=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[i]);
t->data=A[i];
t->link=NULL;
p->link=t;
p=t;
}
printf ("The given Singly Linked List :  ");
display(first);
return 0;
}
```

**OUTPUT**

```
User Input : 1 2 3 4 5
The given Singly Linked List :  1       2       3       4       5
```

## Delete a given integer from the above linked list

```
#include <stdio.h>
#include <stdlib.h>
//Declaring a  singly list node structure
struct node
{
int data;
struct node *link;
}*first;
//traversing through linked list and displaying its elements
void display(struct node *p)
{
while(p)
{
printf ("%d\t",p->data);
p=p->link;
}
}
//deleting random integer
void deleterandom(struct node *p,int m)
{
  struct node *temp;
  int x=m-1;
while(x--)
{
 temp=p;
 p=p->link;
}
temp->link=p->link;
free(p);
}
//creating SLL using array
int main()
{
int A[5],x;
struct node *p,*t;
p=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[0]);
p->data=A[0];
p->link=NULL;
first=p;
for(int i=1;i<5;i++)
{
t=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[i]);
t->data=A[i];
t->link=NULL;
p->link=t;
p=t;
}
display(first);
printf("/nEnter the integer position to be discarded :");
scanf(" %d/n", &x);
deleterandom(first,x);
return 0;
}
```

**OUTPUT**

```
User Input :1        2       3        4        5
Output :      1        2        3        4        5
Enter the integer position to be discarded :3
              1        2        4        5
```

## Display the contents of the above list after deletion

```
#include <stdio.h>
#include <stdlib.h>
//Declaring a  singly list node structure
struct node
{
int data;
struct node *link;
}*first;
//traversing through linked list and displaying its elements
void display(struct node *p)
{
while(p)
{
printf ("%d\t",p->data);
p=p->link;
}
}
//deleting random integer
void deleteDisplay(struct node *p)
{
  struct node *temp;
while(p)
{
 temp=p;
 printf("%d\t", temp->data);
 first=p;
 p=p->link;
 free(temp);
}
printf("%d\t",p->data);
free(p);
exit(1);
}
//creating SLL using array
int main()
{
int A[5],x;
struct node *p,*t;
p=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[0]);
p->data=A[0];
p->link=NULL;
first=p;
for(int i=1;i<5;i++)
{
t=(struct node*)malloc(sizeof(struct node));
scanf("%d", &A[i]);
t->data=A[i];
t->link=NULL;
p->link=t;
p=t;
}
display(first);
printf ("\n Deleting the SLL and printing its element :
");
deleteDisplay(first);
return 0;
}
```

**OUTPUT**

```
User Input : 1      2      3      4      5
             1      2      3      4      5
 Deleting the SLL and printing its element :
             1      2      3      4      5
```

## Create a doubly linked list of integers

```
#include <stdio.h>
#include <stdlib.h>
struct doublyLinkedList
{
  int data;
  struct doublyLinkedList *next;
  struct doublyLinkedList *prev;
}*first;
//creating doubly linked list
void create(int n)
{
  struct doublyLinkedList *p=first;
  p=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &p->data);
  p->next=NULL;
  p->prev=NULL;
  first=p;
while(n--)
{
  struct doublyLinkedList *t;
  t=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &t->data);
  t->next=NULL;
  t->prev=p;
  p->next=t;
  p=t;
}
}
void display(struct doublyLinkedList *p)
{
while(p)
{
  printf("%d\t", p->data);
  p=p->next;
}
}
int main ()
{
  int n;
  printf("Enter the number of data user want to
enter\n");
  scanf("%d", &n);
  create(n-1);
  display(first);
  return 0;
}
```

**OUTPUT**

```
Enter the number of data user want to enter
User Input : 4
                1       2        3      4
                1        2       3       4
```

## Delete a given integer from the above linked list

```
#include <stdio.h>
#include <stdlib.h>
//declaring DLL structure
struct doublyLinkedList
{
  int data;
  struct doublyLinkedList *next;
  struct doublyLinkedList *prev;
}*first;
//creating doubly linked list using dynamic pointer
void create(int n)
{
  struct doublyLinkedList *p=first;
  p=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &p->data);
  p->next=NULL;
  p->prev=NULL;
  first=p;
while(n--)
{
  struct doublyLinkedList *t;
  t=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &t->data);
  t->next=NULL;
  t->prev=p;
  p->next=t;
  p=t;
}
}
void display(struct doublyLinkedList *p)
{
while(p)
{
  printf("%d\t", p->data);
  p=p->next;
}
}
void deleteRandom(int a)
{
  struct doublyLinkedList *p,*temp=first;
  p=first;
  for(int i=1;i<a;i++)
  {
    temp=p;
    p=p->next;
  }
  temp->next = p->next;
  p->next->prev=temp;
  free(p);
}

int main ()
{
  int x,n;
  printf("Enter the number of data user want to enter
and the position of integer to be deleted\n");
  scanf("%d %d", &n ,&x);
  create(n-1);
  printf ("\n The initial linked list is : ");
  display(first);
  deleteRandom(x);
  printf("\nThe final entry after deletion : ");
  display(first);
  return 0;
}
```

**OUTPUT**

```
Enter the number of data user want to enter and the
position of integer to be deleted
User Input : 3      2
               1      2      3
The initial linked list is : 1     2       3
The final entry after deletion : 1      3
```

## Display the contents of the above list after deletion

```
#include <stdio.h>
#include <stdlib.h>
//declaring DLL structure
struct doublyLinkedList
{
  int data;
  struct doublyLinkedList *next;
  struct doublyLinkedList *prev;
}*first;
//creating doubly linked list using dynamic pointer
void create(int n)
{
  struct doublyLinkedList *p=first;
  p=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &p->data);
  p->next=NULL;
  p->prev=NULL;
  first=p;
while(n--)
{
  struct doublyLinkedList *t;
  t=(struct doublyLinkedList *)malloc(sizeof(struct
doublyLinkedList));
  scanf("%d", &t->data);
  t->next=NULL;
  t->prev=p;
  p->next=t;
  p=t;
}
}
void display(struct doublyLinkedList *p)
{
while(p)
{
  printf("%d\t", p->data);
  p=p->next;
}
}
void delete(struct doublyLinkedList *p)
{
  struct doublyLinkedList *temp=first;
   while(p)
   {
     temp=p;
     p=p->next;
     p->prev=NULL;
     printf ("%d", temp->data);
     first=p;
     free(temp);
   }
printf ("%d",p->data);
free(p);
   }
int main ()
{
  int x,n;
  printf("Enter the number of data user want to enter
\n");
  scanf("%d %d", &n ,&x);
  create(n-1);
  printf ("\n The initial linked list is : ");
  display(first);
  delete(x);
  printf("\nThe final entry after deletion : ");
  return 0;
}
```

**OUTPUT**

```
Enter the number of data user want to enter
User Input :3
                1     2     3
The initial linked list is : 1    2     3
The final entry after deletion : 1   2   3
```

## A stack operation to convert a given infix expression to postfix quivalent, implement the stack using an array.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include<limits.h>
struct stack
{
  int top;
  int capacity;
  int *array;
};
struct stack *createStack(int capacity)
{
  struct stack *s=malloc(sizeof(struct stack));
  if(!s)
  return NULL;
  s->capacity=capacity;
  s->top=-1;
  s->array=malloc(s->capacity*sizeof(char));
  if(!s->array)
  return NULL;
  return s;
}
int isEmpty(struct stack *s)
{
  return (s->top==-1);
}
int isFull(struct stack*s)
{
  return (s->top==s->capacity-1);
}
void push(struct stack *s,char data)
{
  if(isFull(s))
  printf("Stack Overflow\n");
  else
  s->array[++s->top]=data;
}
int pop(struct stack*s)
{
  if(isEmpty(s))
  {
    printf ("Stack is empty\n");
    return INT_MIN;
  }
  else
  return (s->array[s->top--]);
}

int peek(struct stack*s)
{
  if(isEmpty(s))
  {
    return INT_MIN;
  }
  else
  return (s->array[s->top]);
}
int priority(char x)
{
  if(x=='(')
  return 0;
  if(x=='+' || x=='-')
  return 1;
  if(x=='*' || x=='/')
  return 2;
}
int infixToPostfix(char expression[])
{
  char *e,x;
  struct stack *s=createStack(8);
  e=expression;
   while(*e!='\0')
   {
    if(isalnum(*e))
    printf("%c",*e);
    else if(*e=='(')
    push(s,*e);
    else if(*e==')')
    {
      while((x=pop(s))!='(')
      printf("%c",x);
    }
    else
    {
      while(priority(peek(s))>=priority(*e))
      printf("%c",pop(s));
      push(s,*e);
    }
    e++;
  }
    while(!isEmpty(s))
    printf("%c", pop(s));
}

int main ()
{
 infixToPostfix("a*b-(c+d)+e");
  return 0;
}
```

**OUTPUT**

```
ab*cd+-e+
```

## A queue ADT using an array

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct Queue
{
  int front, rear;
  int capacity;
  int size;
  int *array;
};
struct Queue *createQueue(int capacity)
{
  struct Queue *Q=malloc(sizeof(struct Queue));
  if(!Q)
  return NULL;
  Q->capacity=capacity;
  Q->front=Q->rear=-1;
  Q->size=0;
  Q->array=malloc(Q->capacity*sizeof(int));
  if(!Q->array)
  return NULL;
  return Q;
}
int size (struct Queue *Q)
{
  return Q->size;
}
int frontElement(struct Queue *Q)
{
  return Q->array[Q->front];
}
int rearElement(struct Queue *Q)
{
  return Q->array[Q->rear];
}
int isEmpty(struct Queue *Q)
{
  return Q->size= =0;
}
int isFull(struct Queue *Q)
{
  return (Q->size= =Q->capacity);
}
void enQueue(struct Queue *Q, int data)
{
  if(isFull(Q))
  printf ("Queue overflow\n");
  else
  {
    Q->rear=(Q->rear+1)%Q->capacity;
    Q->array[Q->rear]=data;
    if(Q->front= =-1)
    Q->front=Q->rear;
    Q->size+ =1;
  }
}
int deQueue(struct Queue *Q)
{
  int data=INT_MIN;
  if(isEmpty(Q))
  {
    printf ("Queue is Empty\n");
    return data;
  }
  data=Q->array[Q->front];
  if(Q->front= =Q->rear)
  {
    Q->front=Q->rear=-1;
    Q->size=0;
  }
  else
  {
    Q->front=(Q->front+1)%Q->capacity;
    Q->size-=1;
  }
  return data;
}
void deleteQueue(struct Queue *Q)
{
  if(Q)
  {
    if(Q->array)
    free(Q->array);
```

```
        free(Q);
    }
}
int main ()
{
    struct Queue *Q;
    Q=createQueue(4);
    enQueue(Q,1);
    enQueue(Q,2);
    enQueue(Q,3);
    enQueue(Q,4);
    enQueue(Q,5);
    printf ("\n Size of Queue : %d\n", size(Q));
    printf ("Front element : %d\t", frontElement(Q));
    printf ("Rear element : %d", rearElement(Q));
    printf ("\nDequeue element : %d\n", deQueue(Q));
    printf ("Dequeue element : %d\n", deQueue(Q));
    printf ("Dequeue element : %d\n", deQueue(Q));
    printf ("Dequeue element : %d\n", deQueue(Q));
    printf ("Dequeue element : %d\n", deQueue(Q));
    enQueue (Q,15);
    enQueue(Q,100);
    printf ("\n Size of queue : %d\n", size(Q));
    printf ("Front element : %d\t", frontElement(Q));
    printf ("Rear element : %d", rearElement(Q));
    deleteQueue(Q);
    return 0;
}
```

**OUTPUT**

```
Size of Queue       : 4
Front element       : 1        Rear element : 4
Dequeue element     : 1
Dequeue element     : 2
Dequeue element     : 3
Dequeue element     : 4
Queue is Empty
Dequeue element     : -2147483648
Size of queue       : 2
Front element       : 15       Rear element : 100
```

## A queue ADT using linked list respectively

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct node
{
  int data;
  struct node *next;
};
struct queue
{
  struct node *front;
  struct node *rear;
};
struct queue *createQueue()
{
  struct queue *q;
  struct node *temp;
  q=malloc(sizeof(struct queue));
  if(!q)
  return NULL;
  temp=malloc(sizeof(struct node));
  q->front=q->rear=NULL;
  return q;
}
int size(struct queue *q)
{
  struct node *temp=q->front;
  int count=0;
  if(q->front= =NULL && q->rear= =NULL)
  return 0;
  while(temp!=q->rear)
  {
    count+ +;
    temp=temp->next;
  }
  if(temp= =q->rear)
  count+ +;
  return count;
}
int frontElement(struct queue *q)
{
  return q->front->data;
}

int rearElement(struct queue *q)
{
  return q->rear->data;
}
void isEmpty(struct queue *q)
{
  if(q->front= =NULL && q->rear= =NULL)
  printf ("Empty Queue\n");
  else
  printf ("Queue is not empty\n");
}
void enQueue(struct queue *q, int num)
{
  struct node *temp;
  temp=(struct node*)malloc(sizeof(struct node));
  temp->data=num;
  temp->next=NULL;
  if(q->rear= =NULL)
  q->front=q->rear=temp;
  else
  {
    q->rear->next=temp;
    q->rear=temp;
  }
}
void deQueue(struct queue *q)
{
  struct node *temp;
  if(q->front= =NULL)
  printf ("Queue is empty\n");
  else
  {
    temp=q->front;
    q->front=q->front->next;
    if(q->front= =NULL)
    q->rear=NULL;
    printf("Removed Element : %d\n", temp->data);
    free(temp);
  }
```

```
}
void printQueue(struct queue *q)
{
  struct node *temp=q->front;
  if(q->front==NULL && q->rear==NULL)
  printf ("Queue is empty\n");
  while(temp!=NULL)
  {
    printf ("%d", temp->data);
    temp=temp->next;
    if(temp!=NULL)
    printf("-->");
  }
}
void deleteQueue(struct queue *q)
{
  struct node *temp;
  while(q->front)
  {
    temp=q->front;
    printf ("Element being deleted : %d\n",temp-
>data);
    q->front=q->front->next;
    free(temp);
  }
  free(q);
}

int main ()
{
  struct queue *q;
  q=createQueue();
  enQueue(q,1);
  enQueue(q,3);
  enQueue(q,5);
  enQueue(q,7);
  enQueue(q,9);
  enQueue(q,11);
  printQueue(q);
  printf ("\n Size of Queue : %d\n", size(q));
  printf ("Front ELement : %d\n", frontElement(q));
  printf ("Rear Element : %d\n", rearElement(q));
  deQueue(q);
  deQueue(q);
  deQueue(q);
  deQueue(q);
  deQueue(q);
  enQueue(q,13);
  enQueue(q,15);
  printQueue(q);
  printf ("\n Size of Queue : %d\n", size(q));
  printf ("Front ELement : %d\n", frontElement(q));
  printf ("Rear Element : %d\n", rearElement(q));
  deleteQueue(q);
  return 0;
}
```

**OUTPUT**

```
1-->3-->5-->7-->9-->11
 Size of Queue : 6
Front ELement : 1
Rear Element : 11
Removed Element : 1
Removed Element : 3
Removed Element : 5
Removed Element : 7
Removed Element : 9
11-->13-->15
 Size of Queue : 3
Front ELement : 11
Rear Element : 15
Element being deleted : 11
Element being deleted : 13
Element being deleted : 15
```

## Create a binary search tree of characters

```
#include <stdio.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
  char data;
  struct node *left;
  struct node *right;
}*Node;
//Creating root node
struct node *create()
{
  char x;
  //struct node *newnode;
  struct node *newnode=(struct
node*)malloc(sizeof(struct node));
  printf("Enter the data for the nodes (0 for no
data)");
 scanf("%c",&x);
  newnode->data=x;
  if(x= =0)
  {
    printf ("Not entered any data\n");
    return NULL;
  }
  printf("Enter the left child of the rooted data : %c
\n",newnode->data);
  newnode->left=create();
  printf("Enter the right child of the rooted data : %c
\n",newnode->data);
  newnode->right=create();
  return newnode;
}
//Inorder traversing using recursion
```

```
void Inorder(struct node *root)
{
  if(root)
  {
    Inorder(root->left);
    printf ("%c\t",root->data);
    Inorder(root->right);
  }
}
//Postorder traversing using recursion
void postorder(struct node *root)
{
  if(root)
  {
    postorder(root->left);
    postorder(root->right);
    printf ("%d  ",root->data);
  }
}
//Preorder traversing using recursion
void preorder(struct node *root)
{
  if(root)
  {
    printf ("%d  ",root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
int main ()
  {
    //struct node *t,*root;
    struct node *root=create();
    struct node *t=root;
    //traverse(t);
    printf("Preorder Traversal : \n");
    preorder(root);
    printf("Inorder Traversal : \n");
    Inorder(root);
    printf("postorder Traversal : \n");
    postorder(root);
    return 0;
  }
```

**OUTPUT**

```
Enter the data for the nodes (-1 for no data)a
Enter the left child of the rooted data  a      Enter the data for the nodes (-1 for no data)b
Enter the left child of the rooted data b      Enter the data for the nodes (-1 for no data)0
Not entered any data
Enter the right child of the rooted data b      Enter the data for the nodes (-1 for no data)0
Not entered any data
Enter the right child of the rooted data a      Enter the data for the nodes (-1 for no data)c
Enter the left child of the rooted data c      Enter the data for the nodes (-1 for no data)0
Not entered any data
Enter the right child of the rooted data c      Enter the data for the nodes (-1 for no data)0
Not entered any data
Root child : a
Preorder traversal
a  b  c
```

## Traverse the above binary tree recursively in Postorder, Inorder and Preorder

```
#include <stdio.h>
#include<limits.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
  int data;
  struct node *left;
  struct node *right;
}*Node;
int max(int a, int b)
{
  return (a>b)?a:b;
}
struct node *create()
{
  int x;
  //struct node *newnode;
  struct node *newnode=(struct
node*)malloc(sizeof(struct node));
  printf ("Enter the data for the nodes (-1 for no
data)");
  scanf("%d",&x);
  newnode->data=x;
  if(x= =-1)
  {
    printf ("Not entered any data\n");
    return NULL;
  }
  printf("Enter the left child of the rooted data
%d\t",newnode->data);
  newnode->left=create();
  printf("Enter the right child of the rooted data
%d\t",newnode->data);
  newnode->right=create();
  return newnode;
}
```

```
int height(struct node *root)
{
  if(root= =NULL)
  return 0;
  int leftHeight= height(root->left);
  int rightHeight=height(root->right);
  return 1 + max(leftHeight,rightHeight);
}

int main ()
  {
    //struct node *t,*root;
    struct node *root=create();
    struct node *t=root;
    //traverse(t);
    printf ("Root child : %d\n",t->data);
    printf("\nHeight : %d\n", height(root));
    return 0;
  }
```

**OUTPUT**

```
Enter the data for the nodes (-1 for no data)1
Enter the left child of the rooted data 1      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 1      Enter the data for the nodes (-1 for no data)2
Enter the left child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 2      Enter the data for the nodes (-1 for no data)3
Enter the left child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 3      Enter the data for the nodes (-1 for no data)4
Enter the left child of the rooted data 4      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 4      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Root child : 1
Height : 4
```

## Number of nodes in a binary tree

```
#include <stdio.h>
#include<limits.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
  int data;
  struct node *left;
  struct node *right;
}*Node;
int max(int a, int b)
{
  return (a>b)?a:b;
}
struct node *create()
{
  int x;
  //struct node *newnode;
  struct node *newnode=(struct
node*)malloc(sizeof(struct node));
  printf ("Enter the data for the nodes (-1 for no
data)");
 scanf("%d",&x);
  newnode->data=x;
  if(x= =-1)
  {
    printf ("Not entered any data\n");
    return NULL;
  }
  printf("Enter the left child of the rooted data
%d\t",newnode->data);
 newnode->left=create();
  printf("Enter the right child of the rooted data
%d\t",newnode->data);
  newnode->right=create();
  return newnode;
}
int Number(struct node *root)
{
  if(root= =NULL)
  return 0;
  int x=Number(root->left);
  int y=Number(root->right);
  return (1 + x + y);
}
int main ()
 {
   //struct node *t,*root;
   struct node *root=create();
   struct node *t=root;
   //traverse(t);
   printf ("Root child : %d\n",t->data);
   printf ("Number of nodes : %d", Number(root));
   return 0;
 }
```

**OUTPUT**

```
Enter the data for the nodes (-1 for no data)1
Enter the left child of the rooted data 1      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 1     Enter the data for the nodes (-1 for no data)2
Enter the left child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 2     Enter the data for the nodes (-1 for no data)3
Enter the left child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 3     Enter the data for the nodes (-1 for no data)4
Enter the left child of the rooted data 4      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 4     Enter the data for the nodes (-1 for no data)-1
Not entered any data
Root child : 1
Number of nodes : 4
```

## Number of Leaf and Non-Leaf Nodes in a Binary Tree

```
#include <stdio.h>
#include<limits.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
  int data;
  struct node *left;
  struct node *right;
}*Node;

int max(int a, int b)
{
  return (a>b)?a:b;
}
/*
struct Queue
{
  int front, rear;
  int capacity;
  int size;
  int *array;
};
struct Queue *createQueue(int capacity)
{
  struct Queue *Q=malloc(sizeof(struct Queue));
  if(!Q)
  return NULL;
  Q->capacity=capacity;
  Q->front=Q->rear=-1;
  Q->size=0;
  Q->array=malloc(Q->capacity*sizeof(int));
  if(!Q->array)
  return NULL;
  return Q;
}
int size (struct Queue *Q)
{
  return Q->size;
```

```
}
int frontElement(struct Queue *Q)
{
  return Q->array[Q->front];
}
int rearElement(struct Queue *Q)
{
  return Q->array[Q->rear];
}
int isEmpty(struct Queue *Q)
{
  return Q->size= =0;
}

int isFull(struct Queue *Q)
{
  return (Q->size= =Q->capacity);
}
void enQueue(struct Queue *Q, int data)
{
  if(isFull(Q))
  printf ("Queue overflow\n");
  else
  {
    Q->rear=(Q->rear+1)%Q->capacity;
    Q->array[Q->rear]=data
    if(Q->front= =-1)
    Q->front=Q->rear;
    Q->size+ =1;
  }
}
int deQueue(struct Queue *Q)
{
  int data=INT_MIN;
  if(isEmpty(Q))
  {
    printf ("Queue is Empty\n");
    return data;
  }
  data=Q->array[Q->front];
  if(Q->front= =Q->rear)
  {
    Q->front=Q->rear=-1;
    Q->size=0;
  }
  else
  {
    Q->front=(Q->front+1)%Q->capacity;
    Q->size-=1;
  }
  return data;
}

void deleteQueue(struct Queue *Q)
{
  if(Q)
  {
    if(Q->array)
    free(Q->array);
    free(Q);
  }
}
*/
struct node *create()
{
  int x;
  //struct node *newnode;
  struct node *newnode=(struct
node*)malloc(sizeof(struct node));
  printf ("Enter the data for the nodes (-1 for no
data)");
  scanf("%d",&x);
  newnode->data=x;
  if(x= =-1)
  {
    printf ("No data\n");
    return NULL;
  }
  printf("Enter the left child of the rooted data
%d\t",newnode->data);
  newnode->left=create();
  printf("Enter the right child of the rooted data
%d\t",newnode->data);
  newnode->right=create();
  return newnode;
```

```
}
int leafNumber(struct node *root)
{
  int count =0;
  if(root->right= =NULL && root->left= =NULL)
  return count+ =1;
  int x=leafNumber(root->left);
  int y=leafNumber(root->right);
  return x+y;
}
/*
int leafNonRecursive(struct node *root)
{
  if(!root)
  return 0;
  struct node *temp;
  struct queue *q=createQueue();
  enQueue(q,root);
  while(!isEmpty(q))
  {
    temp=deQueue(q);
    if(!temp->left && !temp->right)
    count+ +;
    else
    {
      if(temp->left)
      enQueue(q,temp->left);
      if(temp->right)
      enQueue(q,temp->right);
    }
  }
  deleteQueue(q);
  return count;
}
*/
int main ()
  {
    //struct node *t,*root;
    struct node *root=create();
    struct node *t=root;
    //traverse(t);
    printf ("Root child : %d\n",t->data);
```

```
   printf ("Number of leaf nodes : %d",
leafNumber(root));
   return 0;
 }
```

OUTPUT

```
Enter the data for the nodes (-1 for no data)1
Enter the left child of the rooted data 1      Enter the data for the nodes (-1 for no data)2
Enter the left child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
No data
Enter the right child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
No data
Enter the right child of the rooted data 1      Enter the data for the nodes (-1 for no data)3
Enter the left child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
No data
Enter the right child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
No data
Root child : 1
Number of leaf nodes : 2
```

## Create a binary search tree of integers

```
#include <stdio.h>
#include<limits.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
  int data;
  struct node *left;
  struct node *right;
}*Node;
int max(int a, int b)
{
  return (a>b)?a:b;
}
struct node *create()
{
  int x;
  //struct node *newnode;
  struct node *newnode=(struct
node*)malloc(sizeof(struct node));
  printf ("Enter the data for the nodes (-1 for no
data)");
  scanf("%d",&x);
  newnode->data=x;
  if(x= =-1)
  {
    printf ("Not entered any data\n");
    return NULL;
  }
  printf("Enter the left child of the rooted data
%d\t",newnode->data);
  newnode->left=create();
  printf("Enter the right child of the rooted data
%d\t",newnode->data);
  newnode->right=create();
  return newnode;
}
//Preorder traversing using recursion
void preorder(struct node *root)
```

```
{
  if(root)
  {
    printf ("%d  ",root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
int main ()
  {
    //struct node *t,*root;
    struct node *root=create();
    struct node *t=root;
    //traverse(t);
    printf ("Root child : %d\n",t->data);
    printf ("Preorder traversal\n");
    preorder(t);
    return 0;
  }
```

**OUTPUT**

```
Enter the data for the nodes (-1 for no data)1
Enter the left child of the rooted data 1      Enter the data for the nodes (-1 for no data)2
Enter the left child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 2      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 1      Enter the data for the nodes (-1 for no data)3
Enter the left child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 3      Enter the data for the nodes (-1 for no data)-1
Not entered any data
Root child : 1
Preorder traversal
1  2  3
```

## Calculate its height

```
#include <stdio.h>
#include<limits.h>
#include<stdlib.h>
#include <ctype.h>
//declaration of the tree element right, left and data
type
struct node
{
 int data;
 struct node *left;
 struct node *right;
}*Node;

int max(int a, int b)
{
 return (a>b)?a:b;
}
struct node *create()
{
 int x;
 //struct node *newnode;
 struct node *newnode=(struct
node*)malloc(sizeof(struct node));
 printf ("Enter the data for the nodes (-1 for no
data)");
 scanf("%d",&x);
 newnode->data=x;
 if(x==-1)
 {
  printf ("Not entered any data\n");
  return NULL;
 }
 printf("Enter the left child of the rooted data
%d\t",newnode->data);
 newnode->left=create();
 printf("Enter the right child of the rooted data
%d\t",newnode->data);
 newnode->right=create();
 return newnode;
}
```

```
int height(struct node *root)
{
 if(root= =NULL)
  return 0;
 int leftHeight= height(root->left);
 int rightHeight=height(root->right);
 return 1 + max(leftHeight,rightHeight);
}
int main ()
 {
   //struct node *t,*root;
   struct node *root=create();
   struct node *t=root;
   //traverse(t);
   printf ("Root child : %d\n",t->data);
   printf("\nHeight : %d\n", height(root));
   return 0;
 }
```

**OUTPUT**

```
Enter the data for the nodes (-1 for no data)1
Enter the left child of the rooted data 1     Enter the data for the nodes (-1 for no data)2
Enter the left child of the rooted data 2     Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 2     Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 1     Enter the data for the nodes (-1 for no data)3
Enter the left child of the rooted data 3     Enter the data for the nodes (-1 for no data)-1
Not entered any data
Enter the right child of the rooted data 3     Enter the data for the nodes (-1 for no data)-1
Not entered any data
Root child : 1
Height : 2
```

## Insertion Sort

```
import random

def insertionSort(a,n):
    for i in range(1,n):
        key=a[i];
        j=i
        while (a[j-1]>key and j> =1):
            a[j]=a[j-1];
            j-=1;
        a[j]=key;
    for k in range(n):
      print(a[k],end="\t")


x=list(random.randint(1,50) for i in range(10))
n=len(x);
insertionSort(x,n);
```

**OUTPUT**

```
2     9     18    23    30    33    38    41    47    49
50
```

## Merge Sort

```
#include <stdio.h>


void merge(int a[],int temp[],int left,int mid,int right)
{
 int i,left_end,size,temp_pos;
 left_end=mid-1;
 temp_pos=left;
 size=right-left+1;
 while((left< =left_end)&&(mid< =right))
  {
   if(a[left]<a[mid]){
     temp[temp_pos]=a[left];
     temp_pos+ =1;left+ =1;
   }
   else{
     temp[temp_pos]=a[mid];
     temp_pos+ =1;mid+ =1;
   }
  }
  while (left< =left_end){
   temp[temp_pos]=a[left];
   temp_pos+ =1;left+ =1;
  }
  while(mid< =right){
   temp[temp_pos]=a[mid];
   temp_pos+ =1;mid+ =1;
  }
  for (i=0;i< =size;i+ +){
   a[right]=temp[right];
   right-=1;
  }
}


void mergeSort(int a[],int temp[],int left, int right)
{
  int mid;
  if(right>left){
   mid=(left+right)/2;
   mergeSort(a,temp,left,mid);
```

```
    mergeSort(a,temp,mid + 1,right);
    merge(a,temp,left,mid + 1,right);
  }
}


void main()
{
   int  a[] = {10,9,8,7,6,5,764,5,3,10};
  int n = sizeof(a)/sizeof(a[0]);
  int temp[n];
  mergeSort(a,temp,0,n-1);
  for(int i = 0;i < n;i + + )
  printf(" %d\t",a[i]);
}
```

**OUTPUT**

## Quick Sort

```
#include <stdio.h>

//returns the partition position the the quick sort
int partition(int a[],int low,int high){
  int left,right,pivot_value=a[low];
  int temp;
  left=low;
  right=high;
  while(left<right){
    while(a[left]< =pivot_value)
    left+ +;
    while(a[right]>pivot_value)
    right+ +;
    //swaps the left and right value
    if(left<right){
      temp=a[left];
      a[left]=a[right];
      a[right]=temp;
    }
  }
  //swaps the pivot to the partition position
  a[low]=a[right];
  a[right]=pivot_value;
  return (right);
}

//quick sort algorithm for sorting array of integers
void quickSort(int a[], int low , int high ){
  int pivot;
  if(low<high){
    pivot=partition(a,low,high);//finds the partition
position in the array
    quickSort(a,low,pivot-1);
    quickSort(a,pivot+1,high);
  }
  //prints the sorted list
  for(int i=0;i<high;i++)
  printf("%d\t",a[i]);
}


void main(){
     int a[]={5,8,7,6,764,5,10};
    int n=sizeof(a)/sizeof(a[0]);
     quickSort(a,0,n-1);
}
```

**OUTPUT**

```
    5       5       6       7       8       10     764
```