**GREEN UNIVERSITY OF BAN LADESH**

**G**

## Date of Submiss n

**io**

01/12/17

Course title :

LAB REP RT

Data Structures Lab

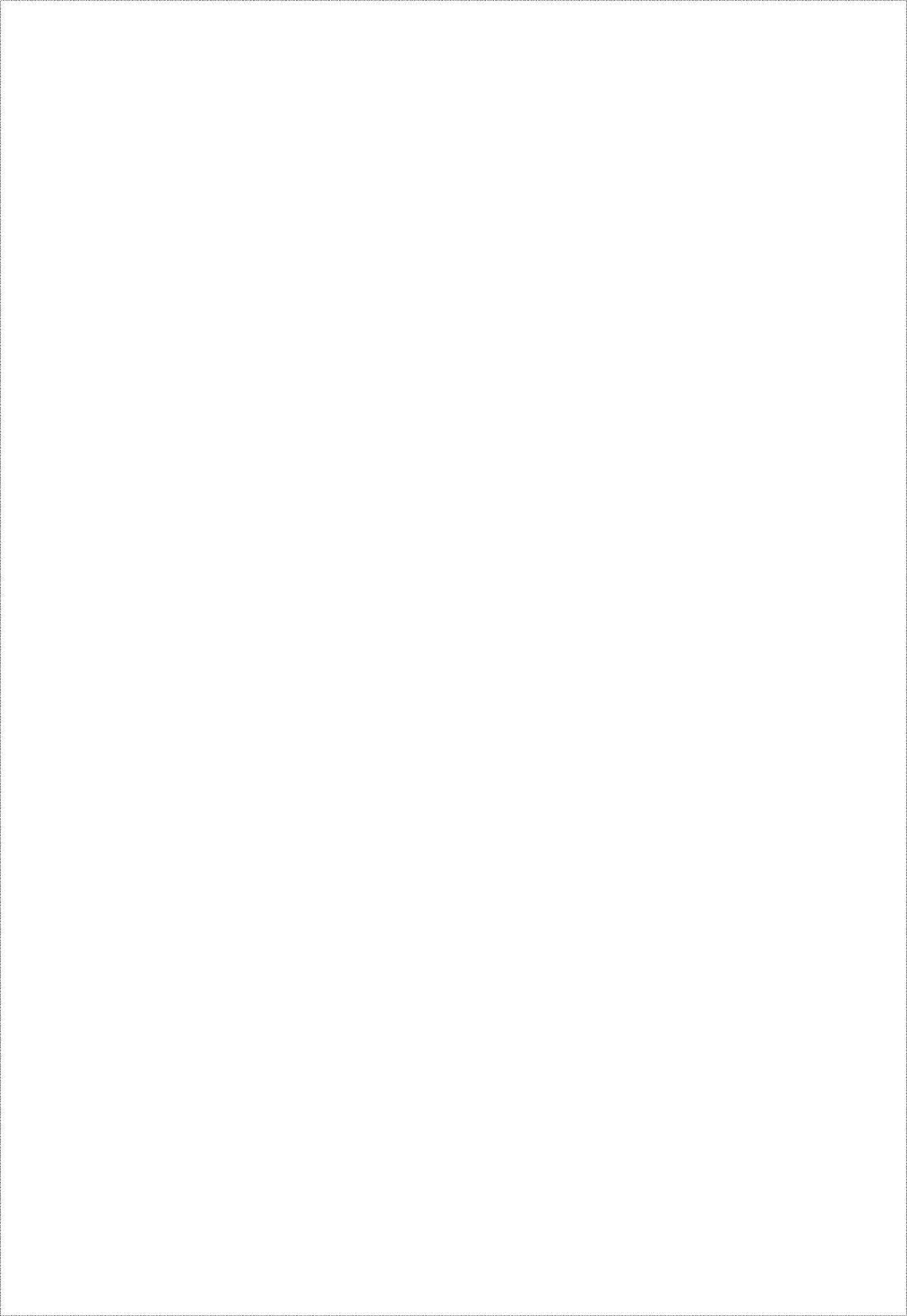
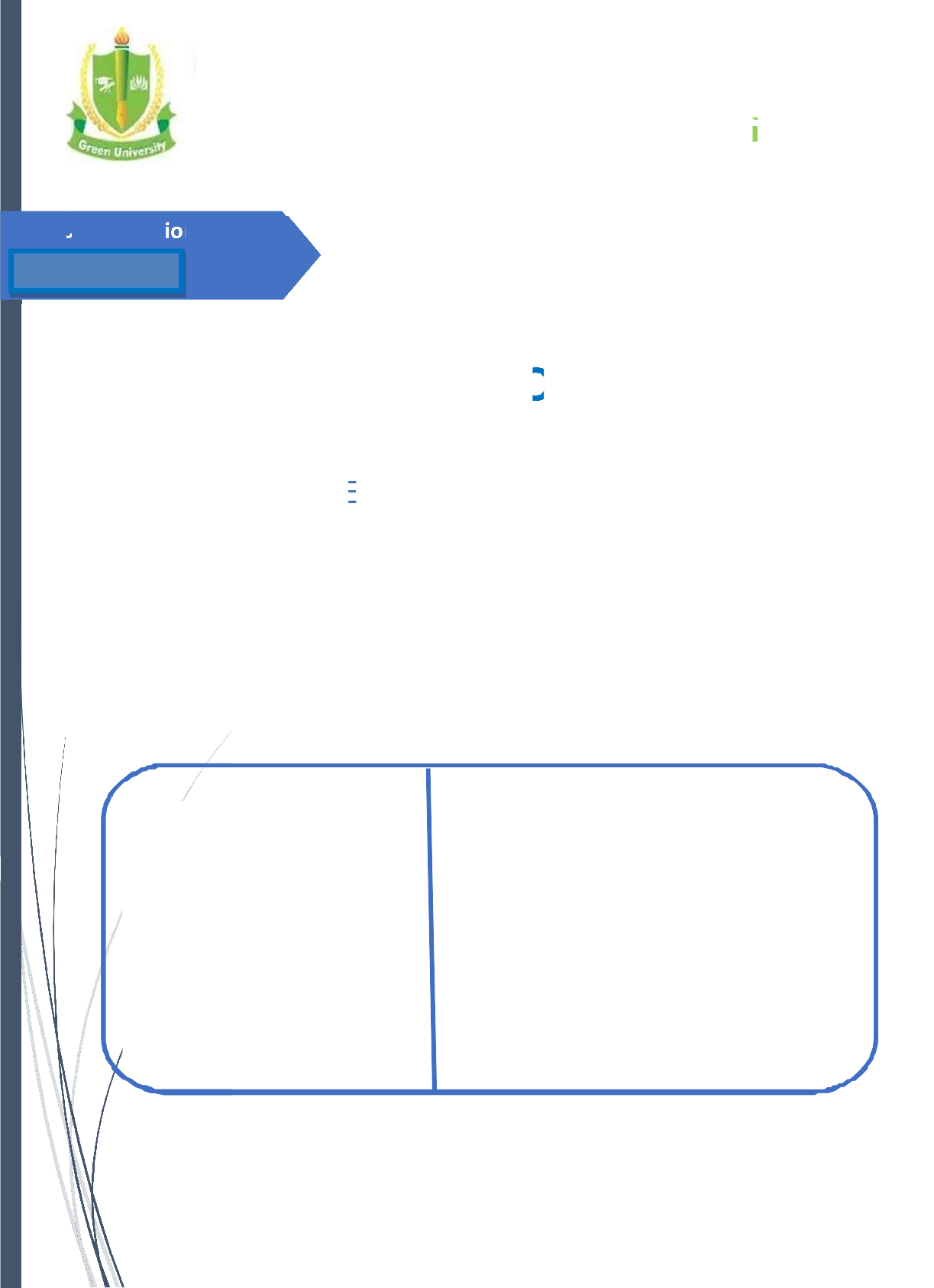
**O**

Course Code : CS -204

E

Date of signature:

Submitted by



Name:

ID:

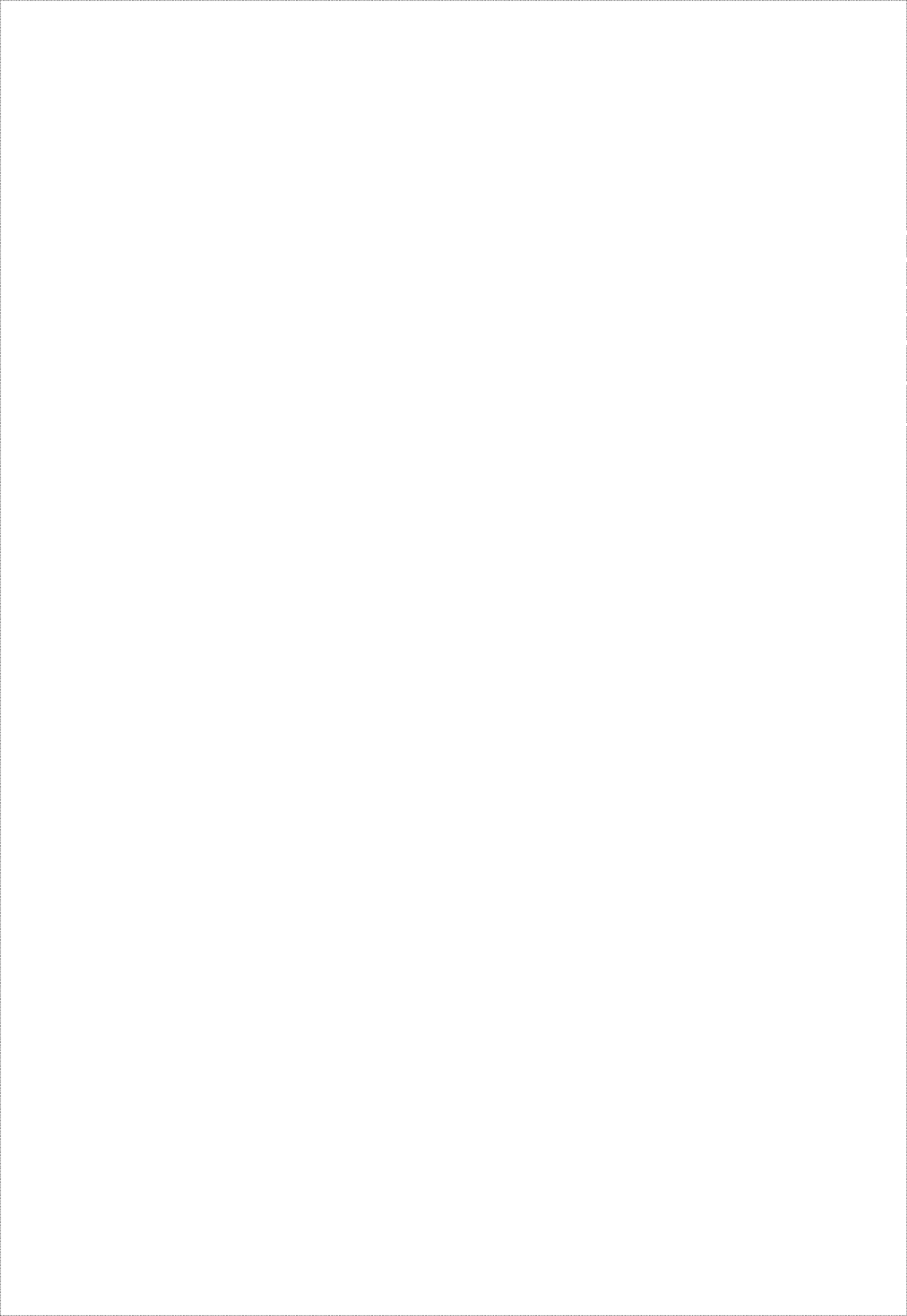
Section:

Green University of BanGladesh

Submitted to

Tamim Al Mahmud Assistant Professor Dept. of CSE

Green University of BanGladesh



|  |  |  |  |
| --- | --- | --- | --- |
|  | | **INDEX** |  |
| **SL** |  |  | **Page** |
| **I** |  | **Data Structures Laboratory: Objectives & Outcomes** | **3** |
|  |  | **PROGRAMMING CODES & OUTPUT** |  |
|  | **1** | **Array Implementation of Stack** | **04** |
|  | **2** | **Implementation of Linear Queue Using Arrays** | **10** |
|  | **3** | **Search an Element in Array Using Binary Search** | **23** |



# DATA STRUCTURES LABORATORY

## OBJECTIVE:

The objective of this lab is to teach students various data structures and to explain them algorithms for performing various operations on these data structures. This lab complements the data structures course. Students will gain practical knowledge by writing and executing programs in C using various data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables and search trees.

## OUTCOMES:

Upon the completion of Data Structures practical course, the student will be able to:

1. **Design** and analyze the time and space efficiency of the data structure.
2. **Identity** the appropriate data structure for given problem.
3. **Understand** the applications of data structures.
4. **Choose** the appropriate data structure and algorithm design method for a specified application.
5. **Understand** which algorithm or data structure to use in different scenarios.
6. **Understand** and apply fundamental algorithmic problems including Tree traversals, Graph traversals.
7. **Compare** different implementations of data structures and to recognize the advantages and disadvantages of them.
8. **Write** complex applications using structured programming methods.



**EXPERIMENT 1: Array Implementation of Stack**

### OBJECTIVE

Implementation of STACK. Convert a given infix expression into its postfix Equivalent, Implement the stack using an array**.**

### RESOURCE:

C++

### PROGRAM LOGIC

1. Create a stack
2. Read an infix expression
3. convert infix exression into postfix expression

### PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

### SOURCE CODE:

**Program to convert a given infix expression into its postfix Equivalent, Implement the**

**stack using an array.**

#include<stdio.h> #include<string.h> #include<stdlib.h> #define MAX 20 char stack[MAX];

int top=1;

char pop(); **/\*declaration of pop function\*/**

void push(char item); **/\*declaration of push function\*/** int prcd(char symbol) **/\*checking the precedence\*/**



{

switch(symbol) **/\*assigning values for symbols\*/**

{

case '+':

case '-': return 2; break;

case '\*':

case '/': return 4;

break;

case '^':return 6; break;

case '(':

case ')':

case '#':return 1; break;

}

}

int(isoperator(char symbol)) **/\*assigning operators\*/**

{

switch(symbol)

{

case '+':

case '\*':

case '-':

case '/':

case '^':

case '(':



case ')':return 1; break; default:return 0;

}

}

### /\*converting infix to postfix\*/

void convertip(char infix[],char postfix[])

{

int i,symbol,j=0; stack[++top]='#'; for(i=0;i<strlen(infix);i++)

{

symbol=infix[i];

if(isoperator(symbol)==0)

{

postfix[j]=symbol; j++;

}

else

{

if(symbol=='(')

push(symbol); **/\*function call for pushing elements into the stack\*/**

else if(symbol==')')

{

while(stack[top]!='(')

{

postfix[j]=pop();



j++;

}

### pop(); /\*function call for popping elements into the stack\*/

}

else

{

if(prcd(symbol)>prcd(stack[top])) push(symbol);

else

{

while(prcd(symbol)<=prcd(stack[top]))

{

postfix[j]=pop();

j++;

}

push(symbol);

### }/\*end of else loop\*/

}**/\*end of else loop\*/**

} **/\*end of else loop\*/** }**/\*end of for loop\*/**

While (stack[top]!='#')

{

postfix[j]=pop(); j++;

}

postfix[j]='\0'; **/\*null terminate string\*/**



}

### /\*main program\*/

void main()

{

char infix[20],postfix[20];

printf("enter the valid infix string \n"); gets(infix);

convertip(infix,postfix); **/\*function call for converting infix to postfix**

**\*/** printf("the corresponding postfix string is:\n"); puts(postfix);

}

### /\*push operation\*/

void push(char item)

{

top++; stack[top]=item;

}

### /\*pop operation\*/

char pop()

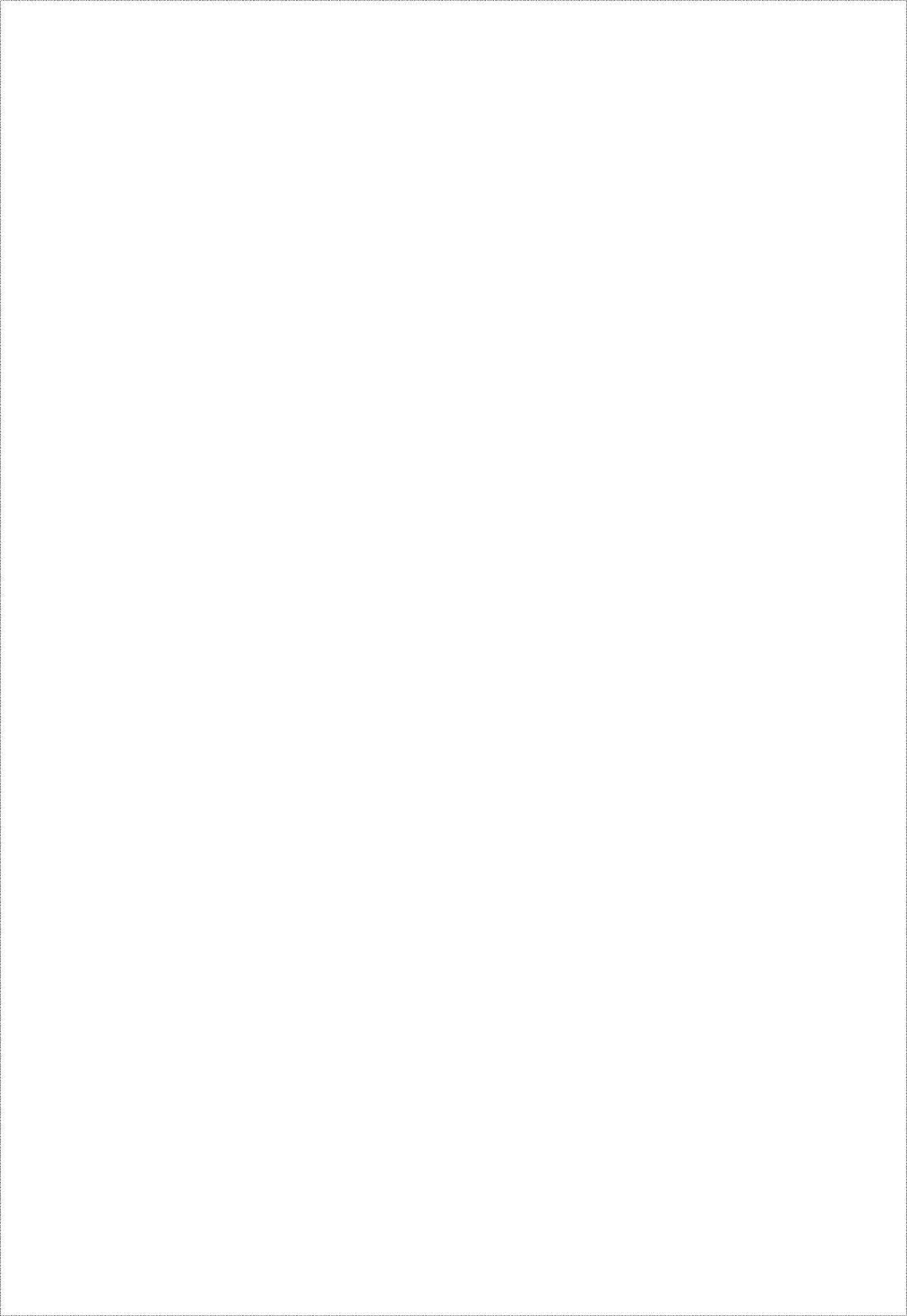
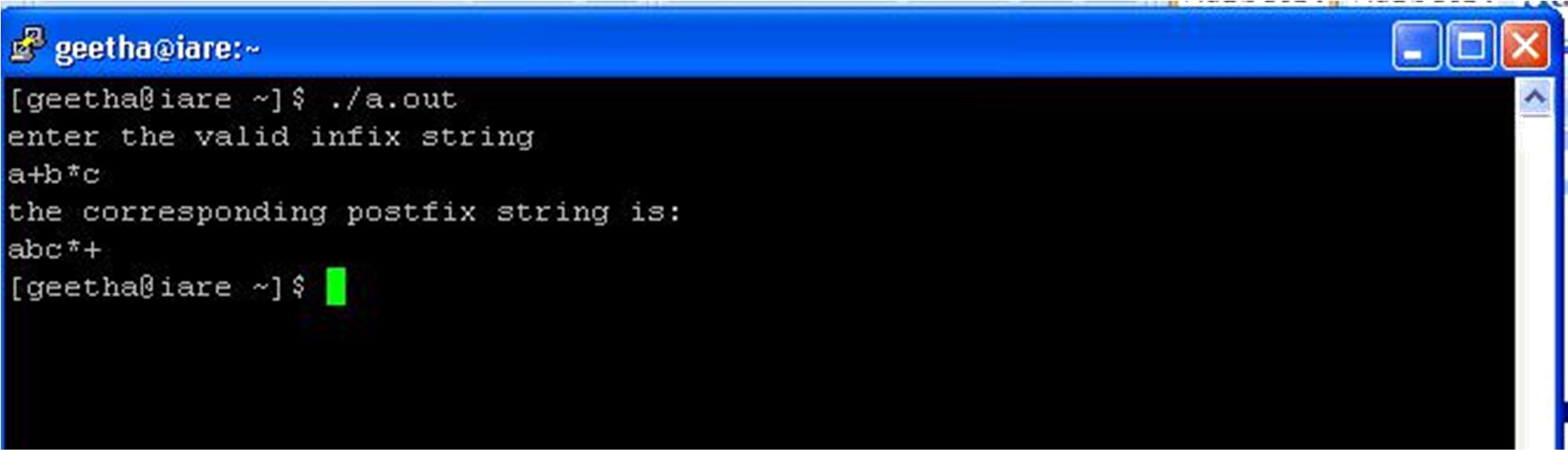
{

char a; a=stack[top]; top--;

return a;

### PRE LAB QUESTIONS

* + 1. what is an expression
    2. what are infix, prefix and postfix notations
    3. what are polish and reverse polish notations



* + 1. which data structure is used for infix to postfix conversion

### LAB ASSIGNMENT

* + 1. Convert the infix expression (a+b)-(c\*d) into post fix form?
    2. Convert the following expression A + (B \* C) - ((D \* E + F) / G) into post form.

### POST LAB QUESTIONS

* + 1. how to represent stack
    2. why reverse polish notation is required
    3. can we evaluate polish notation

### INPUT AND OUTPUT



**EXPERIMENT 2: Implementation of Linear Queue Using Arrays**

### OBJECTIVE

1. To implement a double ended queue ADT using arrays.
2. To implement a double ended queue ADT using doubly linked list.

### RESOURCE:

Turbo C

### PROGRAM LOGIC

**Double ended queue ADT using arrays**

* + 1. Create a linked list
    2. Perform all the operation of double ended queue using arrays
    3. Display the content of queue at last.

### Double ended queue ADT using doubly linked list

1. Create a linked list
2. Perform all the operation of double ended queue using linked list
3. Display the content of queue at last.

### PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

### SOURCE CODE:

**Programs to implement a double ended queue ADT using arrays**

#include<stdio.h> #define SIZE 30 int dequeue[SIZE];

int front=-1,rear=-1; **/\* initializing front and rear\*/**

void insertrear(int);



void deletefront(); void insertfront(int);

void deleterear(); void traverse();

### /\*main program\*/

void main()

{

int choice,item; char ch;

do

{

printf("\n options are");

printf("\n press 1 to insert at rear"); printf("\n press 2 to delete at front");

printf("\n press 3 to insert at front"); printf("\n press 4 to delete at rear"); scanf("%d",&choice);

switch(choice) **/\*switch case\*/**

{

case 1: printf("\n enter the element:"); scanf("%d",&item);

insertrear(item); **/\*function call for inserting element at rear\*/** break;

case 2: deletefront(); **/\*function call for deleting element at front\*/**

break;

case 3: printf("enter the element:");



scanf("%d",&item);

insertfront(item); **/\*function call for inserting element at front\*/** break;

case 4: deleterear();**/\*function call for deleting element at rear\*/** break;

case 5: traverse(); **/\*traversing the list\*/**

break;

default : printf("wrong choice");

### } /\*end of switch case\*/

printf("\n do you want to perform more operations?(Y/N):");

fflush(stdin); scanf(" %c",&ch);

} while(ch=='Y'||ch=='y');

}

### /\*insertion at rear\*/

void insertrear(int value) **/\*function definition\*/**

{

if(rear==(SIZE-1))

{

printf("overflow"); return;

}

else

{

if(front==-1)

{



printf("underflow so front will be modified"); front=front+1;

}

rear=rear+1; dequeue[rear]=value;

}

}

### /\*deletion at front\*/

void deletefront() **/\*function definition\*/**

{

int value; if(front==-1)

{

printf("queue is already empty");

value=-1;

}

else

{

value=dequeue[front];

if(front==rear)

{

printf("queue contains only one item"); rear=-1;

front=-1;

}

else

front=front+1;



printf("removed element from front is %d",value);

}

### /\*insertion at front\*/

void insertfront(int value) **/\*function definition\*/**

{

if(front==0)

{

printf("front is at the beginning"); printf("here insertion is not possible");

return;

}

else

{

if(front==-1)

{

printf("queue is empty so both pointers will modified");

front=front+1; rear=rear+1;

}

else

{

front=front-1;

}

dequeue[front]=value;

}

### /\*deletion at rear\*/

void deleterear() **/\*function definition\*/**

{

int value; if(front==-1)

{

}

else

{

printf("queue is already empty"); return;

value=dequeue[rear]; if(rear==front)

{

printf("queue contains only one item"); printf("rear and front will be modified");

rear=-1; front=-1;

}

else

{

rear=rear-1;

}

}

printf("\n the removed element from rear is:%d",value);

}

### /\*traverse operation\*/

void traverse() **/\*function definition\*/**



{

int i; if(front==-1)

{

printf("queue empty"); return;

}

else

{

printf("\n value in the queue are as follow:"); for(i=front;i<=rear;i++) printf("\n%d",dequeue[i]);

}

}

### program to implement double ended queue adt using doubly linked list

#include <stdio.h> #include <stdlib.h>

### /\*declaring a structure to create a node\*/

struct node

{

int data;

struct node \*prev, \*next;

};

struct node \*head = NULL, \*tail = NULL; struct node \* createNode(int data)

{



### /\*allocating implicit memory to the node\*/

struct node \*newnode = (struct node \*)malloc(sizeof (struct node));

newnode->data = data;

newnode->next = newnode->prev = NULL; return (newnode);

}

### /\* create sentinel(dummy head & tail) that helps us to do insertion and deletion operation at front and rear so easily. And these dummy head and tail wont get deleted till the end of execution of this program \*/

void createSentinels() **/\*creating a head and tail\*/** {

head = createNode(0); tail = createNode(0); head->next = tail;

tail->prev = head;

}

### /\* insertion at the front of the queue \*/

void enqueueAtFront(int data)

{

struct node \*newnode, \*temp; newnode = createNode(data); temp = head->next;

head->next = newnode; newnode->prev = head;

newnode->next = temp; temp->prev = newnode;

}

### /\*insertion at the rear of the queue \*/

void enqueueAtRear(int data)

{

struct node \*newnode, \*temp; newnode = createNode(data);

temp = tail->prev;

tail->prev = newnode; newnode->next = tail; newnode->prev = temp; temp->next = newnode;

}

### /\* deletion at the front of the queue \*/

void dequeueAtFront()

{

struct node \*temp;

if (head->next == tail)

{

}

Else

{

}

printf("Queue is empty\n");

temp = head->next;

head->next = temp->next; temp->next->prev = head;

free(temp);

return;}

### /\* deletion at the rear of the queue \*/

void dequeueAtRear()

{

struct node \*temp;

if (tail->prev == head)

{

}

Else

{

printf("Queue is empty\n");



temp = tail->prev;

tail->prev = temp->prev; temp->prev->next = tail; free(temp);

}

return;

}

### /\* display elements present in the queue \*/

void display()

{

struct node \*temp;

if (head->next == tail)

{

printf("Queue is empty\n"); return;

}



temp = head->next; while (temp != tail)

{

printf("%-3d", temp->data); temp = temp->next;

}

printf("\n");

}

### /\*main program\*/

int main()

{

int data, ch;

createSentinels(); while (1)

{

printf("1. Enqueue at front\n2. Enqueue at rear\n"); printf("3. Dequeue at front\n4. Dequeue at rear\n");

printf("5. Display\n6. Exit\n"); printf("Enter your choice:");

scanf("%d", &ch);

switch (ch) **/\*switch case\*/**

{

case 1:

printf("Enter the data to insert:"); scanf("%d", &data); enqueueAtFront(data);



break;

case 2:

printf("Enter ur data to insert:"); scanf("%d", &data); enqueueAtRear(data);

break;

case 3:

dequeueAtFront(); break;

case 4:

dequeueAtRear(); break;

case 5:

display(); break;

case 6:

exit(0); default:

printf("Pls. enter correct option\n"); break;

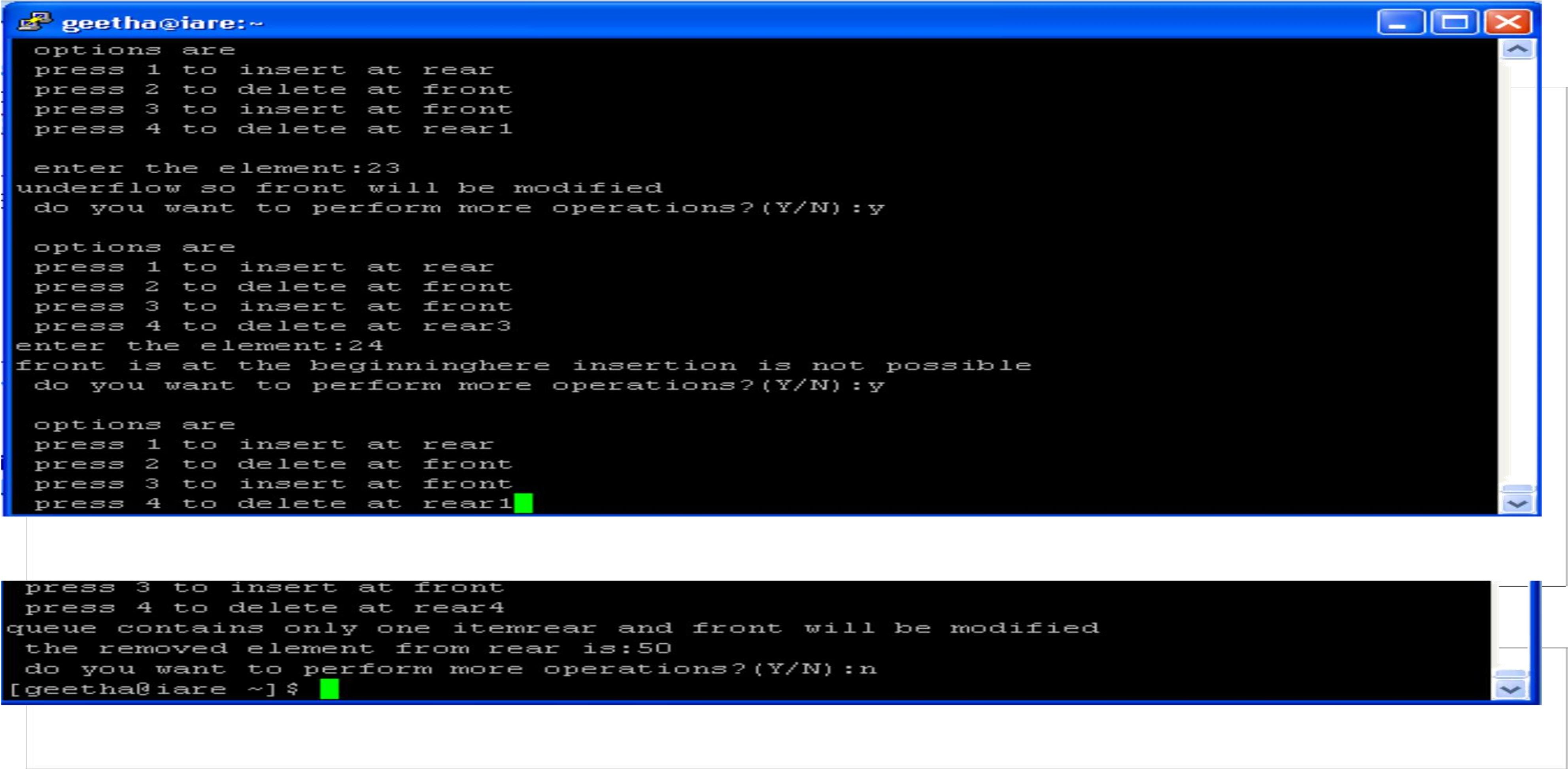
### } /\*end of switch case\*/

}

return 0;

}

22



### PRE LAB QUESTIONS

* + 1. what is queue and its operations
    2. what is double ended queue
    3. differentiate queue and double ended queue

### LAB ASSIGNMENT

* + 1. Write a program to insert an element when rear is at last position
    2. Write a program to delete an element when front is at last position

### POST LAB QUESTIONS

* + 1. Write the condition for queue full
    2. Write the condition for queue empty
    3. List the advantages of double ended queue over queue

### INPUT AND OUTPUT

**A double ended queue ADT using arrays**

**double ended queue adt using doubly linked list**



**EXPERIMENT 3: Search an Element in Array Using Binary Search**

### OBJECTIVE

1. To create a binary search tree of characters.
2. Traverse the above Binary search tree recursively in Post order.

### RESOURCE:

Turbo C/C++

### PROGRAM LOGIC

1. Create binary tree with the property binary search tree
2. Visit the tree in post order
3. Visit in the order left, right, root
4. Display the visited nodes

### PROCEDURE:

Go to debug -> run or press CTRL + F9 to run the program

### SOURCE CODE:

**/\*program for creating and traversing the binary search tree\*/**

#include<stdio.h>

#include<stdlib.h> typedef struct BST

{

char d;

### /\*declaring a structure to create a node\*/

struct BST \*lc,\*rc;

}node;

### /\*main program\*/

void main()

{

int choice; char ans='N'; int key;

node \*nn,\*root,\*parent;

root=NULL;

printf("\n program for binary search tree"); do

{

printf("\n 1.create");

printf("\n 2.resurcive traverse"); printf("\n 3.exit");

printf("\n enter your choice"); scanf("%d",&choice); switch(choice) **/\*switch case\*/**

{

case 1:

do

{

nn=(node \*)malloc(sizeof(node));

printf("\n enter the elements"); nn->lc=NULL;

nn->rc=NULL;

scanf(" %c",&nn->d); if(root==NULL) root=nn;

else

insert(root,nn);

printf("\n want to enter more elements?(Y/N)");

case 2:

scanf(" %c",&ans);

}while(ans=='y'); break;

if(root==NULL)

printf("tree is not created"); else

{

printf("\n the inorder display:"); inorder(root);

printf("\n the preorder display:"); preorder(root);

printf("\n the postorder display:"); postorder(root);

}

break;

### } /\*end of switch case\*/



}while(choice!=3);

}

### /\*insertion operation\*/

void insert(node \*root,node \*nn)

{

int c,d; c=nn->d; d=root->d; if(c<d)

{

if(root->lc==NULL) root->lc=nn;

else

insert(root->lc,nn);

}

}

### /\*inorder traversal\*/

void inorder(node \*temp)

{

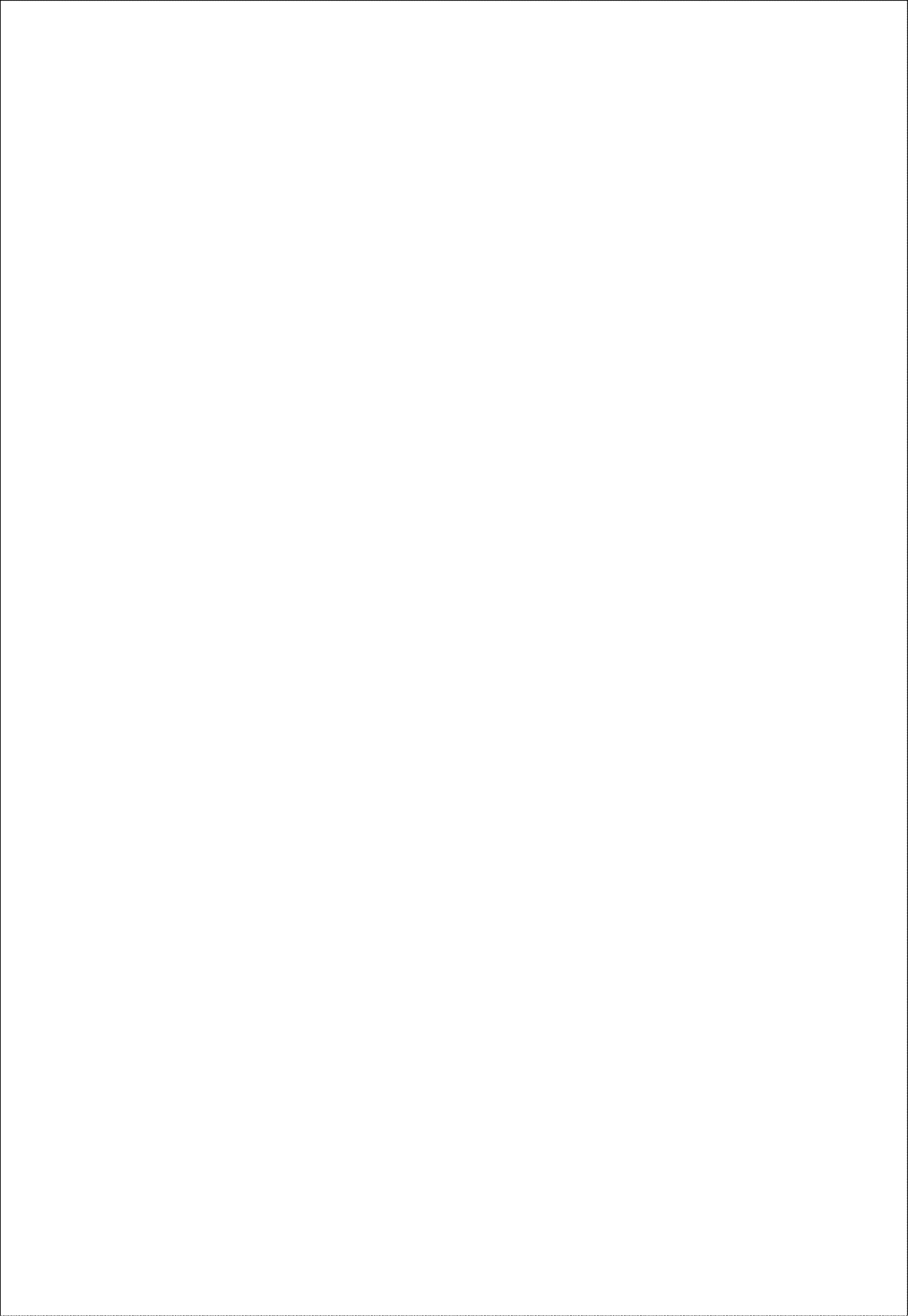
if(temp!=NULL)

{

inorder(temp->lc); printf(" %c",temp->d); inorder(temp->rc);

}

}



### /\*preorder traversal\*/

void preorder(node \*temp)

{

if(temp!=NULL)

{

printf(" %c",temp->d);

preorder(temp->lc); preorder(temp->rc);

}

}

### /\*postorder traversal\*/

void postorder(node \*temp)

{

if(temp!=NULL)

{

postorder(temp->lc); postorder(temp->rc); printf(" %c",temp->d);

}

}

### PRE LAB QUESTIONS

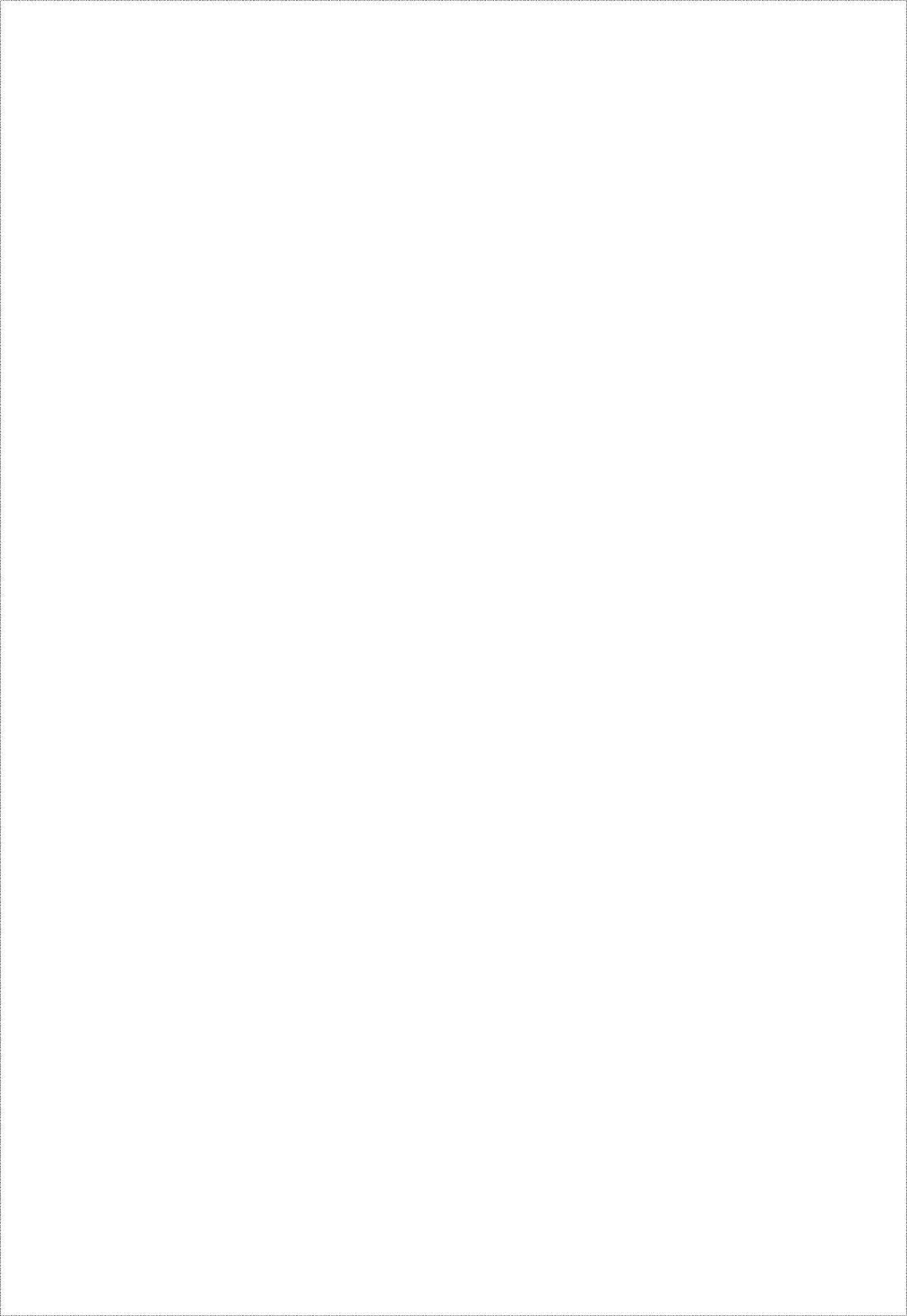
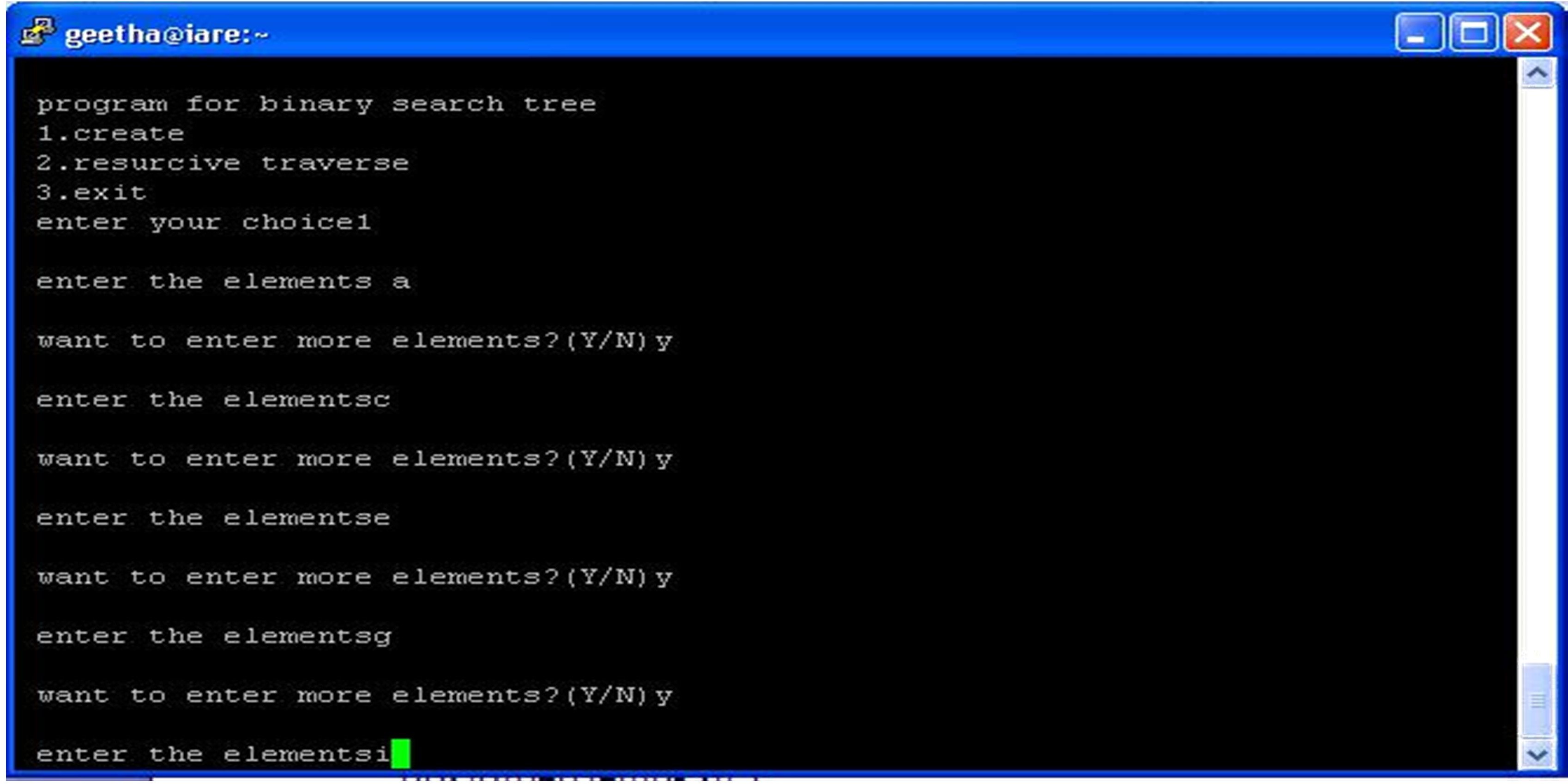
* + 1. Differentiate between BST and complete BST
    2. What are the properties of BST
    3. How many nodes will be there in given nth level.

### LAB ASSIGNMENT

* + 1. Construct a binary search tree for the following 80, 40, 75, 30, 20, 90, 50

### POST LAB QUESTIONS

* + 1. List the various tree traversal techniques are there
    2. Write the necessary condition for inserting element into BST



* + 1. List the applications of BST

### INPUT AND OUTP