G.S. Mandal's

Maharashtra Institute of Technology, Aurangabad

Department of Computer Science and Engineering

LAB MANUAL

CSE205: Data Structures

(2019-20 Part 1)

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Vision

To develop the department as a center of excellence in the field of computer science and engineering by imparting knowledge & training to the students for meeting growing needs of the industry & society.

Mission

Providing quality education through a well-designed curriculum in tune with the challenging needs of software industry by providing state of the art facilities and to impart knowledge in the thrust areas of computer science and engineering.

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Pogram Educational Objectives

PEO1: To prepare the students to achieve success in Computing Domain to create individual careers, innovations or to work as a key contributor to the private or Government sector and society.

PEO2: To develop the ability among the students to understand Computing and mathematical fundamentals and apply the principles of Computer Science for analyzing, designing and testing software for solving problems.

PEO3: To empower the students with ability to quickly reflect the changes in the new technologies in the area of computer software, hardware, networking and database management.

PEO4: To promote the students with awareness for lifelong learning, introduce them to professional practice, ethics and code of professionalism to remain continuous in their profession and leaders in technological society.

Program Specific Objectives

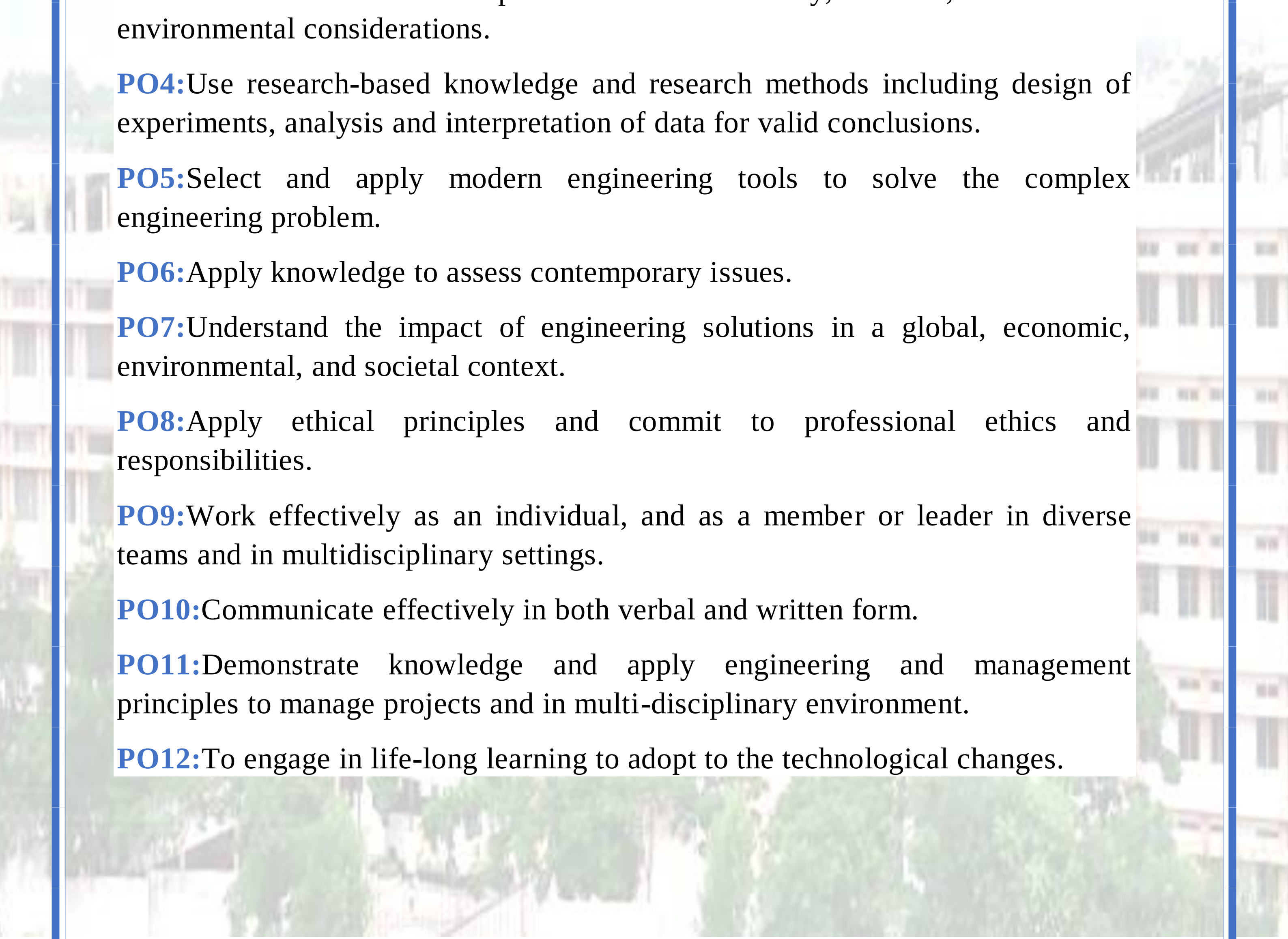
PSO1: Identify appropriate data structures and algorithms for a given contextual problem and develop programs to design and implement web applications.

PSO3: Design and manage the large databases and develop their own databases to solve real world problems and to design, build, manage networks and apply wireless techniques in mobile based applications.

PSO3: Design a variety of computer-based components and systems using computer hardware, system software, systems integration process and use standard testing tools for assuring the software quality.

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Program Outcomes



PO1: Apply knowledge of mathematics, science, and engineering fundamentals to solve problems in Computer science and Engineering.

PO2: Identify, formulate and analyze complex problems.

PO3: Design system components or processes to meet the desired needs within realistic constraints for the public health and safety, cultural, societal and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data for valid conclusions.

PO5: Select and apply modern engineering tools to solve the complex engineering problem.

PO6: Apply knowledge to assess contemporary issues.

PO7: Understand the impact of engineering solutions in a global, economic, environmental, and societal context.

PO8: Apply ethical principles and commit to professional ethics and responsibilities.

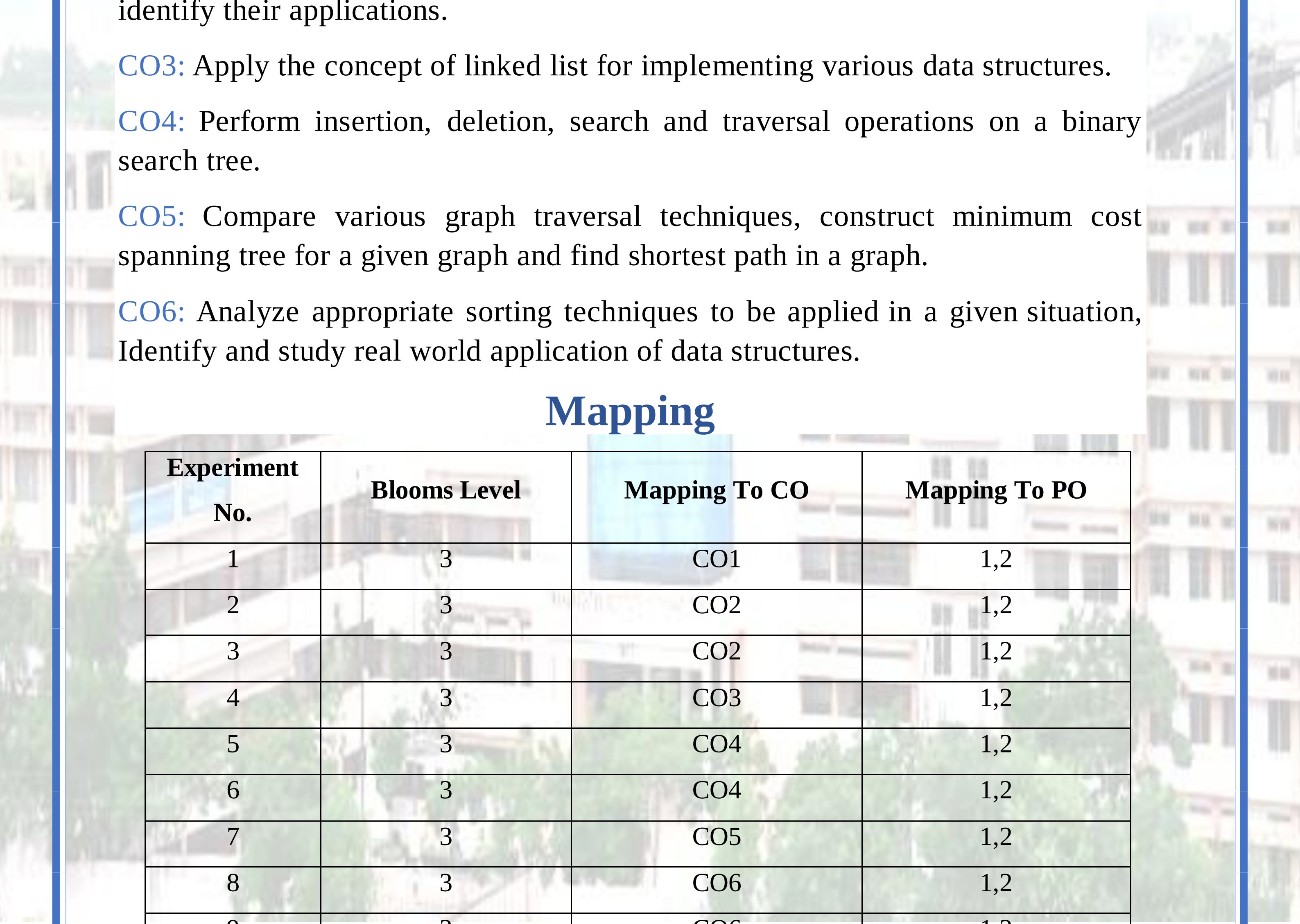
PO9: Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10: Communicate effectively in both verbal and written form.

PO11: Demonstrate knowledge and apply engineering and management principles to manage projects and in multi-disciplinary environment.

PO12: To engage in life-long learning to adopt to the technological changes.

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia



Course: CSE205: Data Structures

Course Outcomes:

After Completing the course students will be able to

CO1: Define the concept of data structures and discriminate the usage of various data structures in approaching a problem.

CO2: Develop static and dynamic implementation of stacks and queues and identify their applications.

CO3: Apply the concept of linked list for implementing various data structures.

CO4: Perform insertion, deletion, search and traversal operations on a binary search tree.

CO5: Compare various graph traversal techniques, construct minimum cost spanning tree for a given graph and find shortest path in a graph.

CO6: Analyze appropriate sorting techniques to be applied in a given situation, Identify and study real world application of data structures.

Mapping

Experiment No.	Blooms Level	Mapping To CO	Mapping To PO
1	3	CO1	1,2
2	3	CO2	1,2
3	3	CO2	1,2
4	3	CO3	1,2
5	3	CO4	1,2
6	3	CO4	1,2
7	3	CO5	1,2
8	3	CO6	1,2
9	3	CO6	1,2
10	3	CO6	1,2

	G.S. MANDAL'S **MAHARASHTRA INSTITUTE OF TECHNOLOGY, AURABGABAD**	
	LAB WORK INSTRUCTION SHEET	
	DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING	
CLASS: S.Y. B. TECH PART: 1 (2019-20)	**LAB: 512-B**	**SUBJECT: CSE205-Data Structures**

Index

	G.S. MANDAL'S MAHARASHTRA INSTITUTE OF TECHNOLOGY, AURABGABAD	
	LAB WORK INSTRUCTION SHEET	
	DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING	
CLASS: S.Y. B. TECH PART: 1 (2019-20)	**LAB: 512-B**	**SUBJECT: CSE205-Data Structures**

Experiment No:1

Aim: To create a banking application using Structures in C.

Theory:

When programming, it is often convenient to have a single name with which to refer to a group of a related values. Structures provide a way of storing many different values in variables of potentially different types under the same name. This makes it a more modular program, which is easier to modify because its design makes things more compact. Structures are generally useful whenever a lot of data needs to be grouped together--for instance, they can be used to hold records from a database or to store information about contacts in an address book. In the contacts example, a structure could be used that would hold all of the information about a single contact--name, address, phone number, and so forth.

structure

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

creating a structure

'struct' keyword is used to create a structure. Following is an example.

```
struct addrress
{
char name[50];
char street[100];
char city[50];
char state[20]
nt pin;
};
```

Declaring structure variables

A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

```
// A variable declaration with structure declaration.
struct Point
{
int x, y;
} p1; // The variable p1 is declared with 'Point'

// A variable declaration like basic data types
struct Point
{
```

```
int x, y;
};

int main()
{
struct Point p1; // The variable p1 is declared like a normal variable
}
```

Array of structures

Like other primitive data types, we can create an array of structures.

```
struct Point
{
int x, y;
};

int main()
{
// Create an array of structures
struct Point arr[10];

// Access array members
arr[0].x = 10;
arr[0].y = 20;

printf("%d %d", arr[0].x, arr[0].y);
return 0;
}
```

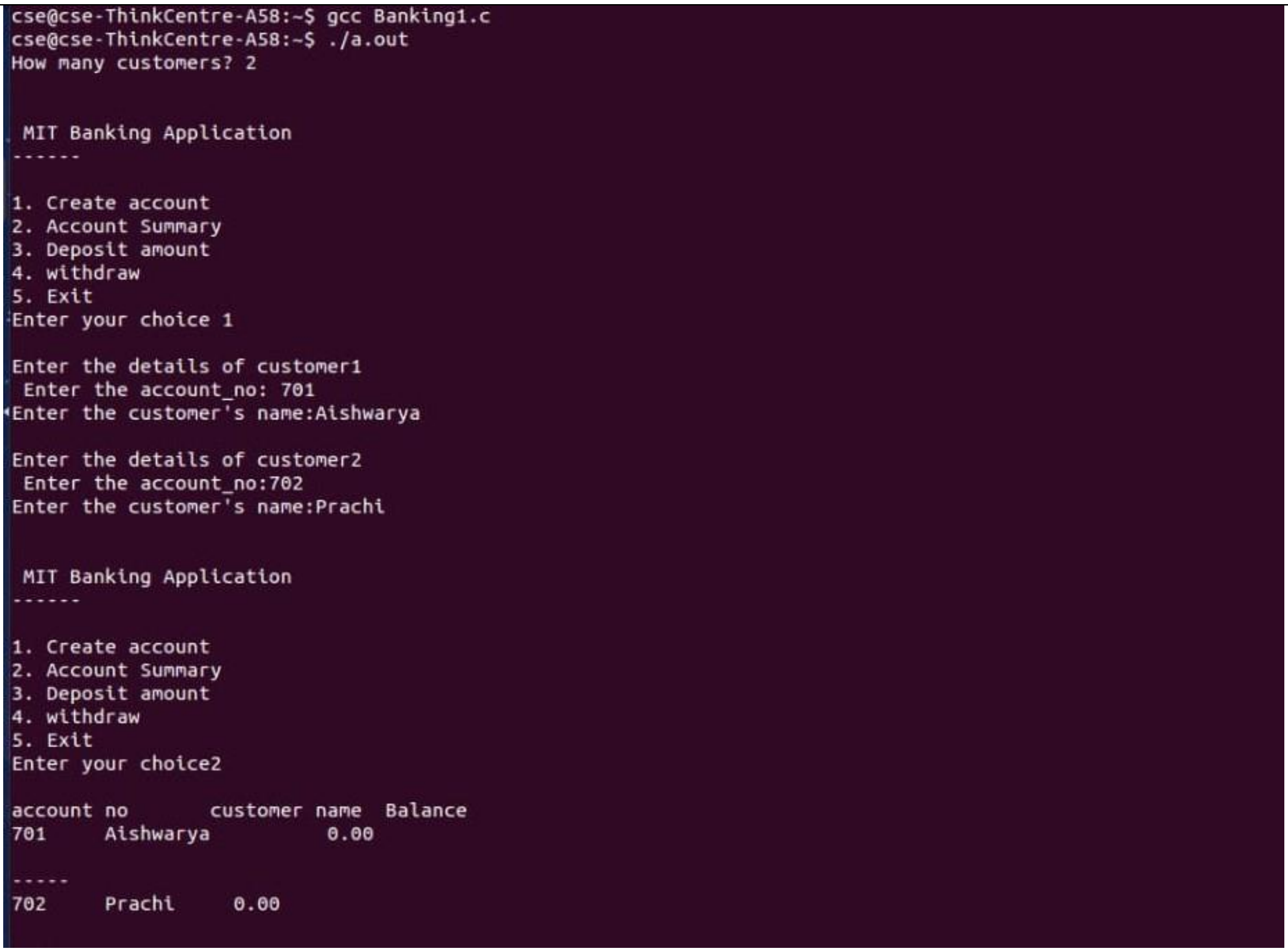
Structures as Function Arguments

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

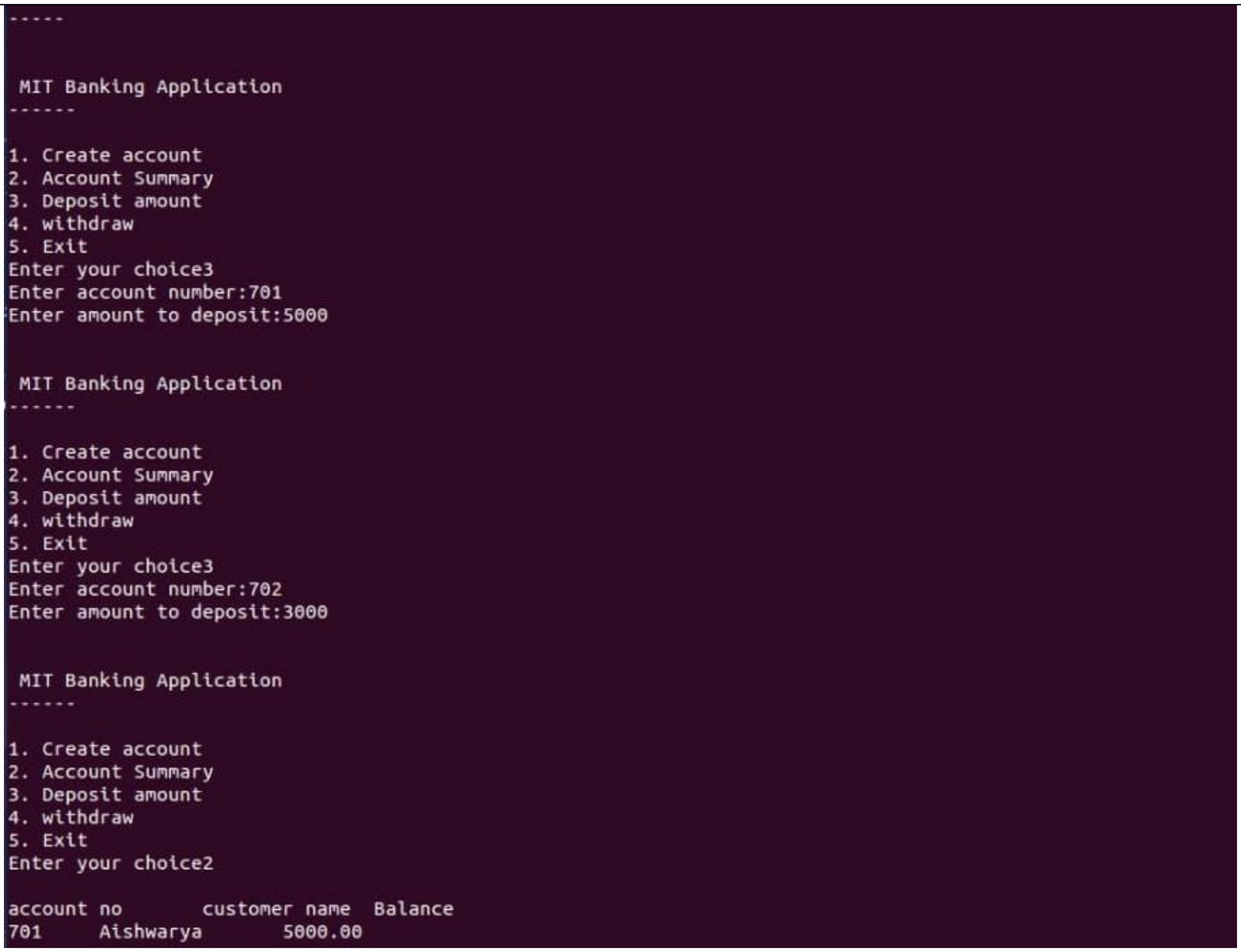
```
#include <stdio.h>
#include <string.h>

struct Books {
char  title[50];
char  author[50];
char  subject[100];
  int   book_id;
};
Void main()
```

```
{struct Books b1;
printBook(b1);

void printBook( struct Books book ) {

printf( "Book title : %s\n", book.title);
printf( "Book author : %s\n", book.author);
printf( "Book subject : %s\n", book.subject);
printf( "Book book_id : %d\n", book.book_id);
}
```

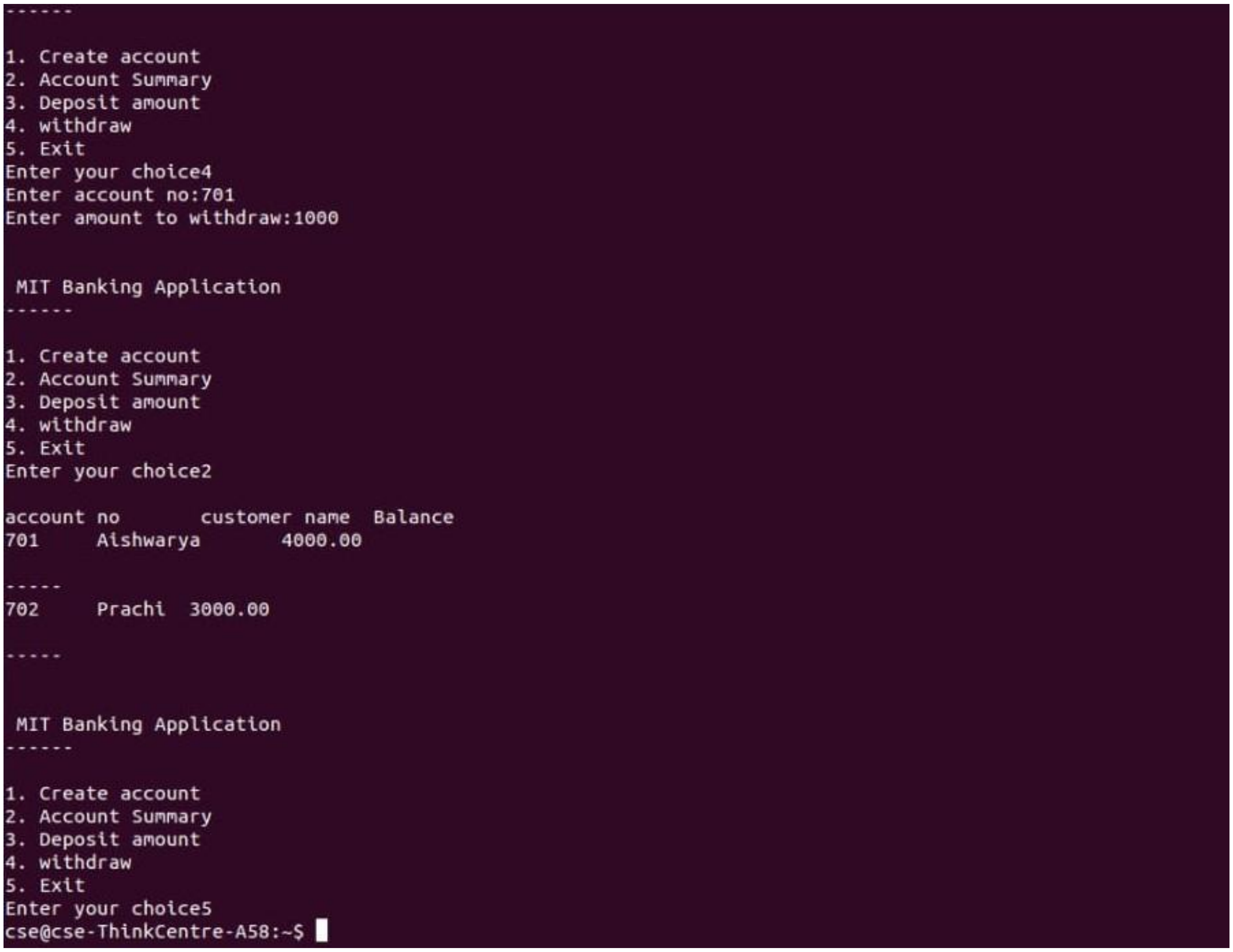
Input/ Sample Expected Output:



```
cse@cse-ThinkCentre-A58:~$ gcc Banking1.c
cse@cse-ThinkCentre-A58:~$ ./a.out
How many customers? 2


 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice 1

Enter the details of customer1
 Enter the account_no: 701
Enter the customer's name:Aishwarya

Enter the details of customer2
 Enter the account_no:702
Enter the customer's name:Prachi


 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice2

account no       customer name   Balance
701     Aishwarya          0.00

-----
702     Prachi     0.00
```



```
-----

 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice3
Enter account number:701
Enter amount to deposit:5000

 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice3
Enter account number:702
Enter amount to deposit:3000

 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice2

account no       customer name   Balance
701      Aishwarya        5000.00
```


```
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice4
Enter account no:701
Enter amount to withdraw:1000


 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice2

account no       customer name  Balance
701     Aishwarya       4000.00

-----
702     Prachi  3000.00

-----


 MIT Banking Application
------

1. Create account
2. Account Summary
3. Deposit amount
4. withdraw
5. Exit
Enter your choice5
cse@cse-ThinkCentre-A58:~$ 
```

CONCLUSION:-Thus the program for MIT BANKING APPLICATION using array of structures was executed successfully and the output was verified.

Experiment No:2

Aim: To develop a program for IMPLEMENTATION OF STACK using an array.

Theory:

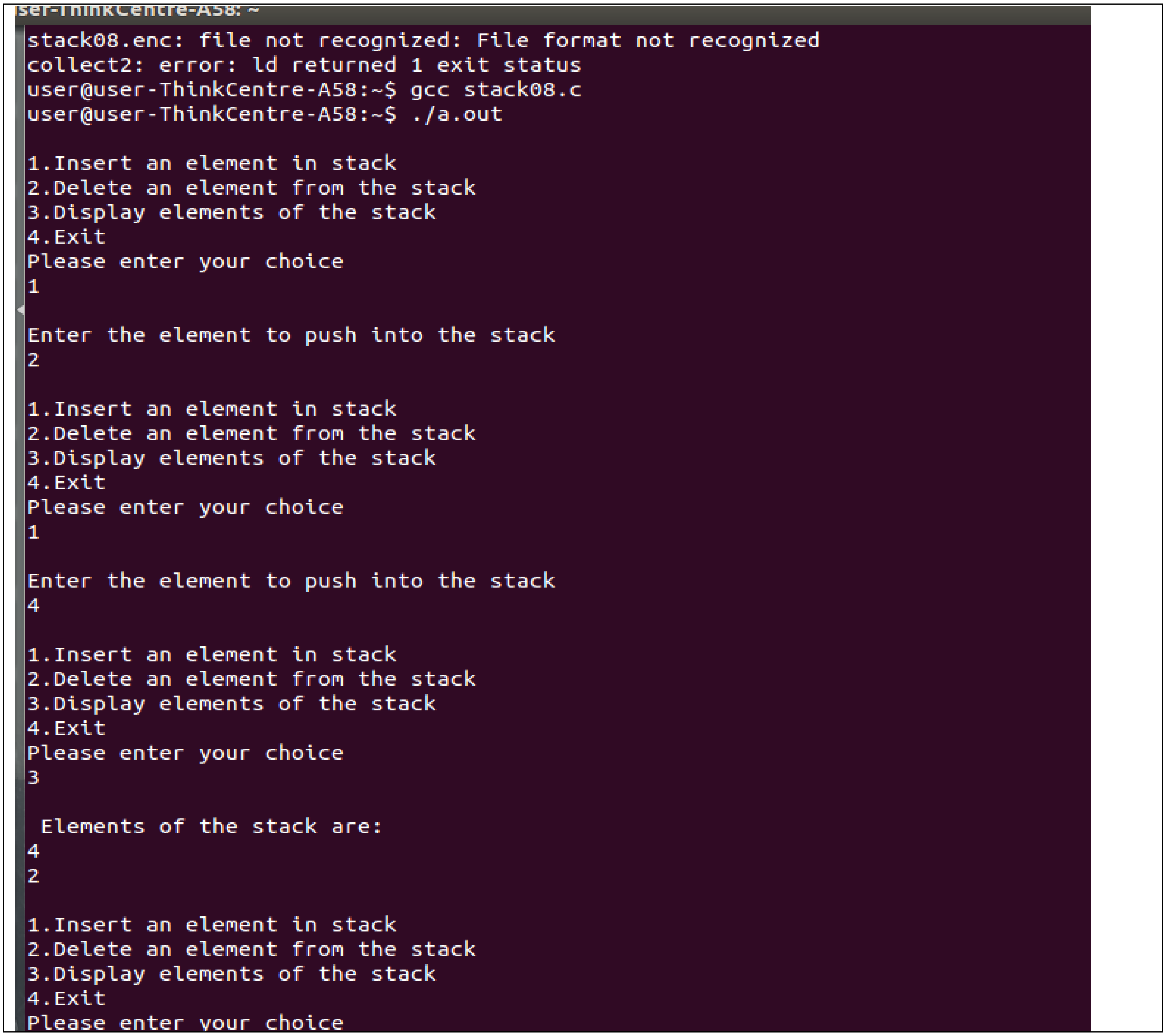
In computer science, a stack is an abstract data type that serves as a collection of elements with two principal operations:
1. **Push:** which adds an element to the stack.
2. **Pop:** which removes the most recently added element that was not yet removed.

The order in which elements come off a stack gives rise to its alternative name,**LIFO(last in,first out).**Additionally, a peek operation may give access to the top without modifying the stack. The name "stack" for this type of structure comes from the analogy to a set of physical items stacked on top of each other, which makes it easy to take an item off the top the stack, while getting to an item deeper in the stack may require taking off multiple other items first.
A stack can be implemented in two ways:
1.Using array i.e. static implementation of stack.
2.Using linked list dynamic implementation of stack.

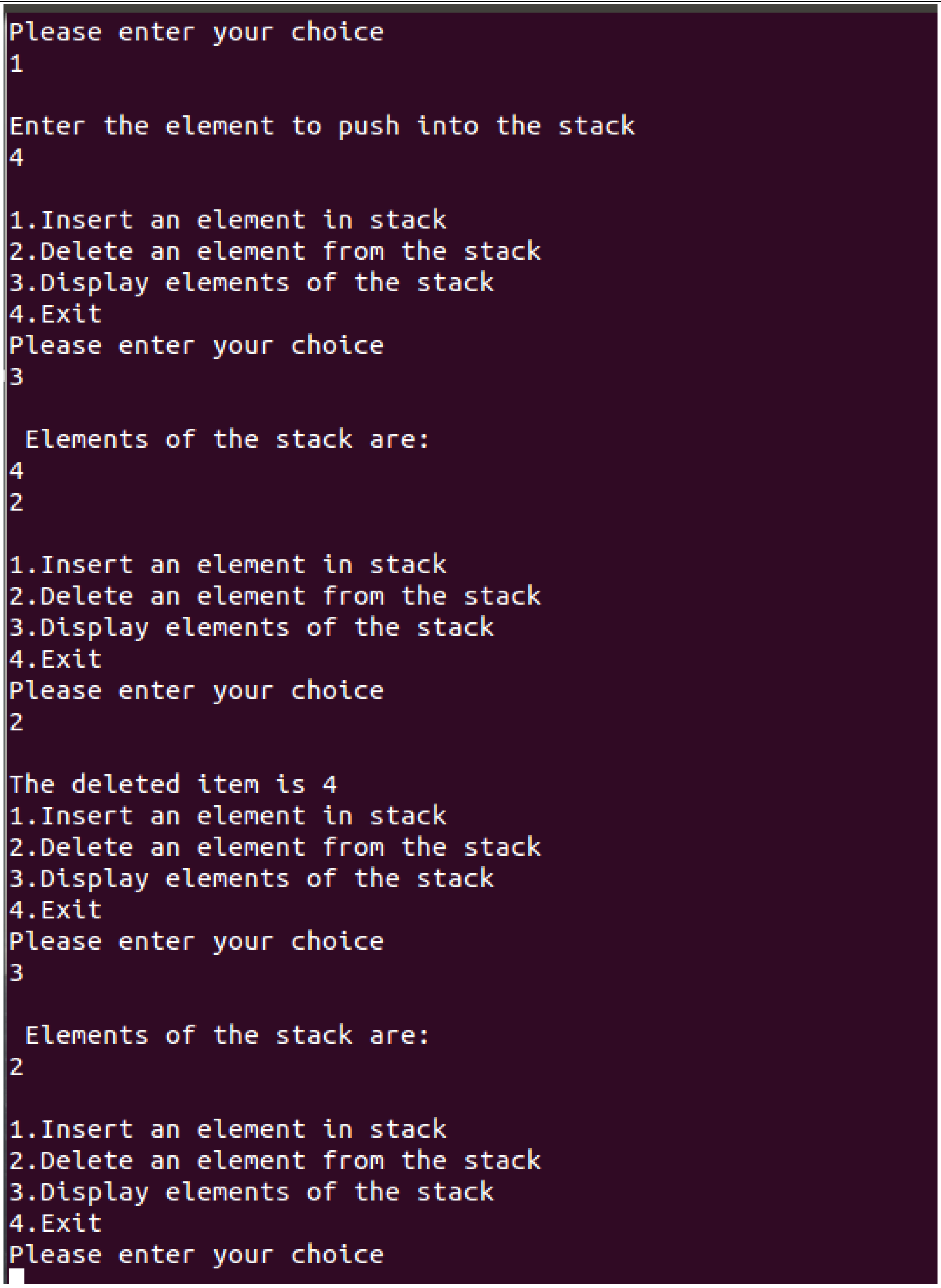
ARRAY IMPLEMENTATION OF STACK:

In an array, elements can be added or deleted at any position but while implementing stack ,we have to permit insertions and deletions at top of the stack only. So we can take a variable top, which keeps the position of the topmost element in an array. Initially when the stack is empty, value of top is initialized to -1.For push operation, first the value of top is increment by -1 and then the new element is pushed at the position of top.For pop operation, first the element at the position of top is popped and the top is decrement by 1.

INPUT / SAMPLE EXPECTED OUTPUT:



```
ser-ThinkCentre-A58: ~
stack08.enc: file not recognized: File format not recognized
collect2: error: ld returned 1 exit status
user@user-ThinkCentre-A58:~$ gcc stack08.c
user@user-ThinkCentre-A58:~$ ./a.out

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
1

Enter the element to push into the stack
2

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
1

Enter the element to push into the stack
4

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
3

 Elements of the stack are:
4
2

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
```



```
Please enter your choice
1

Enter the element to push into the stack
4

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
3

 Elements of the stack are:
4
2

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
2

The deleted item is 4
1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
3

 Elements of the stack are:
2

1.Insert an element in stack
2.Delete an element from the stack
3.Display elements of the stack
4.Exit
Please enter your choice
```

CONCLUSION: Hence, we have learnt the program for implementation of stack and output was verified.

EXPERIMENT NO:3

Aim: To develop a program for IMPLEMENTATION OF QUEUE' using an array

Theory :A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Basic features of Queue

1. Like stack, queue is also an ordered list of elements of similar data types.
2. Queue is a FIFO(First in First Out) structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. peek( ) function is oftenly used to return the value of first element without dequeuing it.
5. enqueue ( ) function is used when elements has to be pushed into the queue.
6. Dequeue ( ) function is used when same element has to be removed from the queue.

Implementation of Queue Data Structure

Queue can be implemented using an array. The easiest way of implementing a queue is by using an array. Initially the front and the **rear** of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the **rear** keeps on moving ahead, always pointing to the position where the next element will be inserted, while the **front** remains at the first index.

Algorithm for ENQUEUE operation

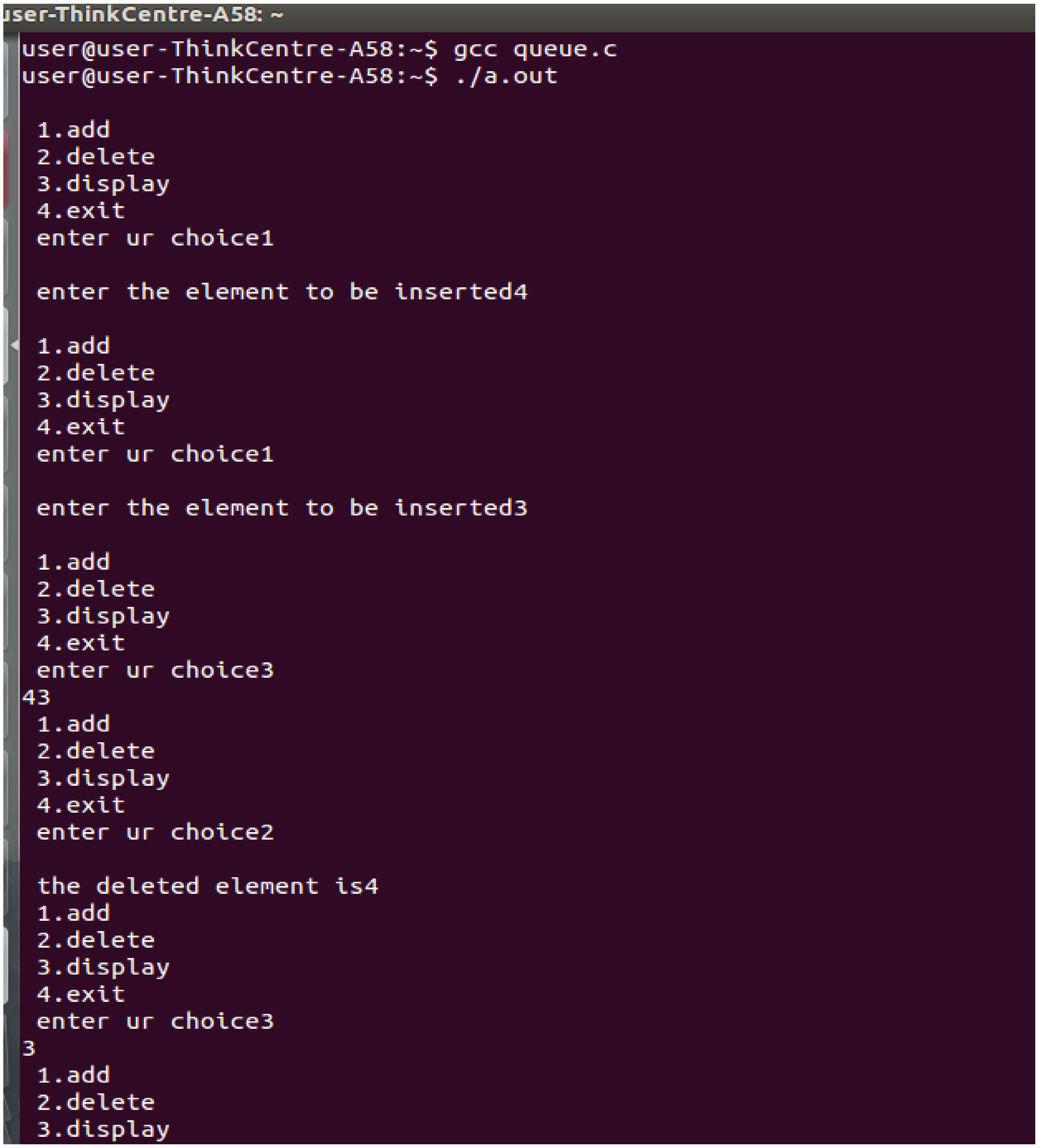
1. Check if the queue is full or not.
2. If the queue is full, then print overflow error and exit the program.
3. If the queue is not full, then increment the rear and add the element.

Algorithm for DEQUEUE operation

1. Check if the queue is empty or not.
2. If the queue is empty, then print underflow error and exit the program.
3. if the queue is not empty, then print the element at the front and increment the front.

CLASS: S.Y. B. TECH PART: 1 (2019-20)	**LAB: 512-B**	**SUBJECT: CSE205-Data Structures**

INPUT/ EXPECTED SAMPLE OUTPUT:



```
user-ThinkCentre-A58: ~
user@user-ThinkCentre-A58:~$ gcc queue.c
user@user-ThinkCentre-A58:~$ ./a.out

 1.add
 2.delete
 3.display
 4.exit
 enter ur choice1

 enter the element to be inserted4

 1.add
 2.delete
 3.display
 4.exit
 enter ur choice1

 enter the element to be inserted3

 1.add
 2.delete
 3.display
 4.exit
 enter ur choice3
43
 1.add
 2.delete
 3.display
 4.exit
 enter ur choice2

 the deleted element is4
 1.add
 2.delete
 3.display
 4.exit
 enter ur choice3
3
 1.add
 2.delete
 3.display
```

CONCLUSION: Thus the program for Queue operations was executed and the output was verified.

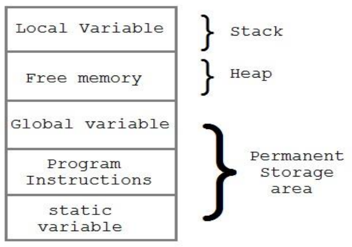
EXPERIMENT NO.4

AIM: To develop a program of basic operation on linked list using C Programming Language- Dynamic Memory Allocation

THEORY:

MEMORY ALLOCATION IN C:

Static memory allocation	**Dynamic memory allocation**
In static memory allocation, memory is allocated while writing the C program. Actually, user requested memory will be allocated at compile time.	In dynamic memory allocation, memory is allocated while executing the program. That means at run time.
Memory size can't be modified while execution. Example: array	Memory size can be modified while execution. Example: Linked list



Memory Allocation Process

Global variables, static variables and program instructions get their memory in **permanent** storage area whereas **local** variables are stored in a memory area called **Stack**.

The memory space between these two region is known as **Heap** area. This region is used for dynamic memory allocation during execution of the program. The size of heap keep changing

DYNAMIC MEMORY ALLOCATION IN C:

The process of allocating memory during program execution is called dynamic memory allocation.

DYNAMIC MEMORY ALLOCATION FUNCTIONS IN C:

C language offers 4 dynamic memory allocation functions. They are,

1. malloc()
2. calloc()
3. realloc()
4. free()

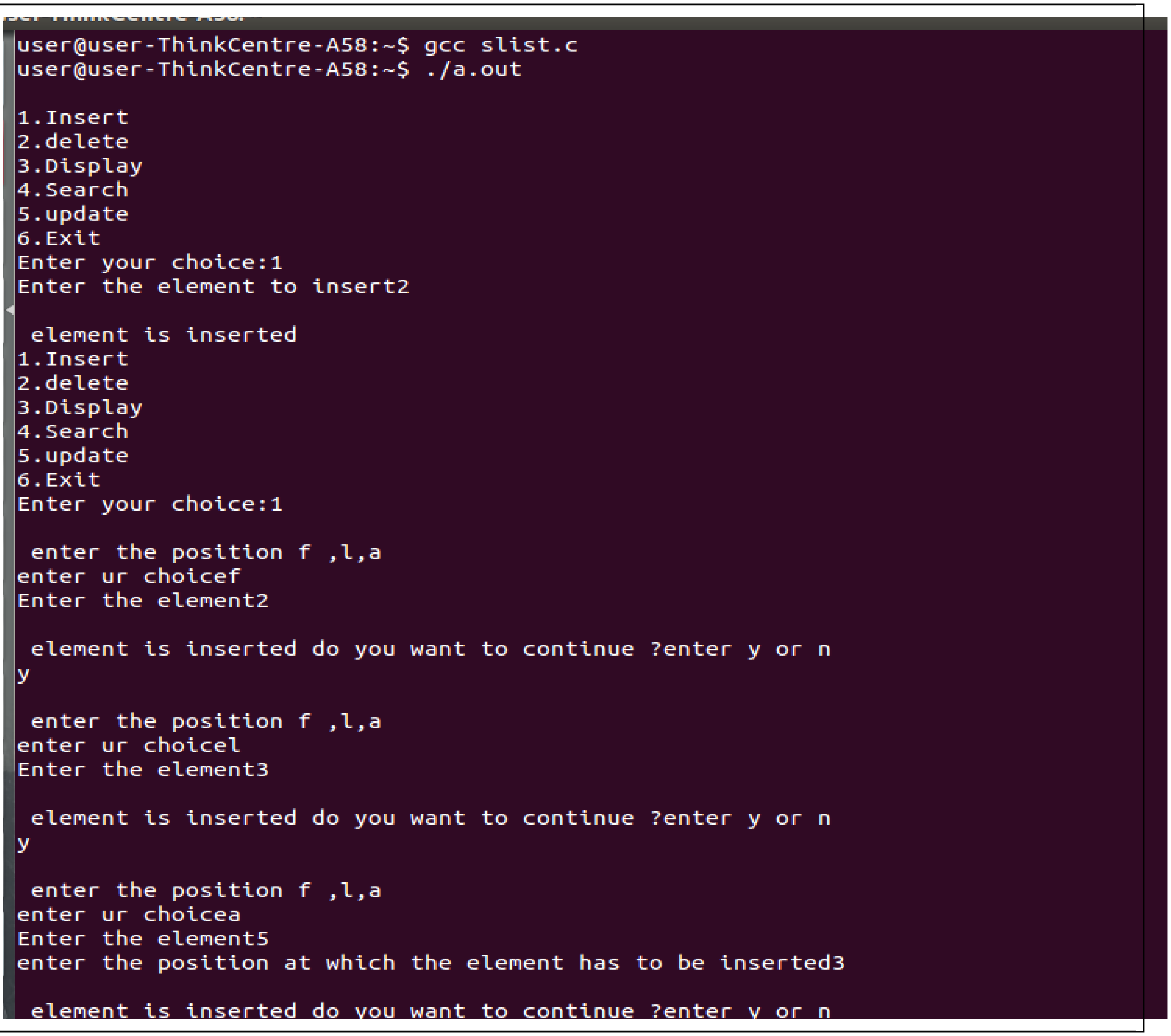
Function	Syntax
malloc ()	malloc (number *sizeof(int));
calloc ()	calloc (number, sizeof(int));
realloc ()	realloc (pointer_name, number * sizeof(int));
free ()	free (pointer_name);

Function	**Description**
malloc()	allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space
calloc()	allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory
free	releases previously allocated memory
realloc	modify the size of previously allocated space

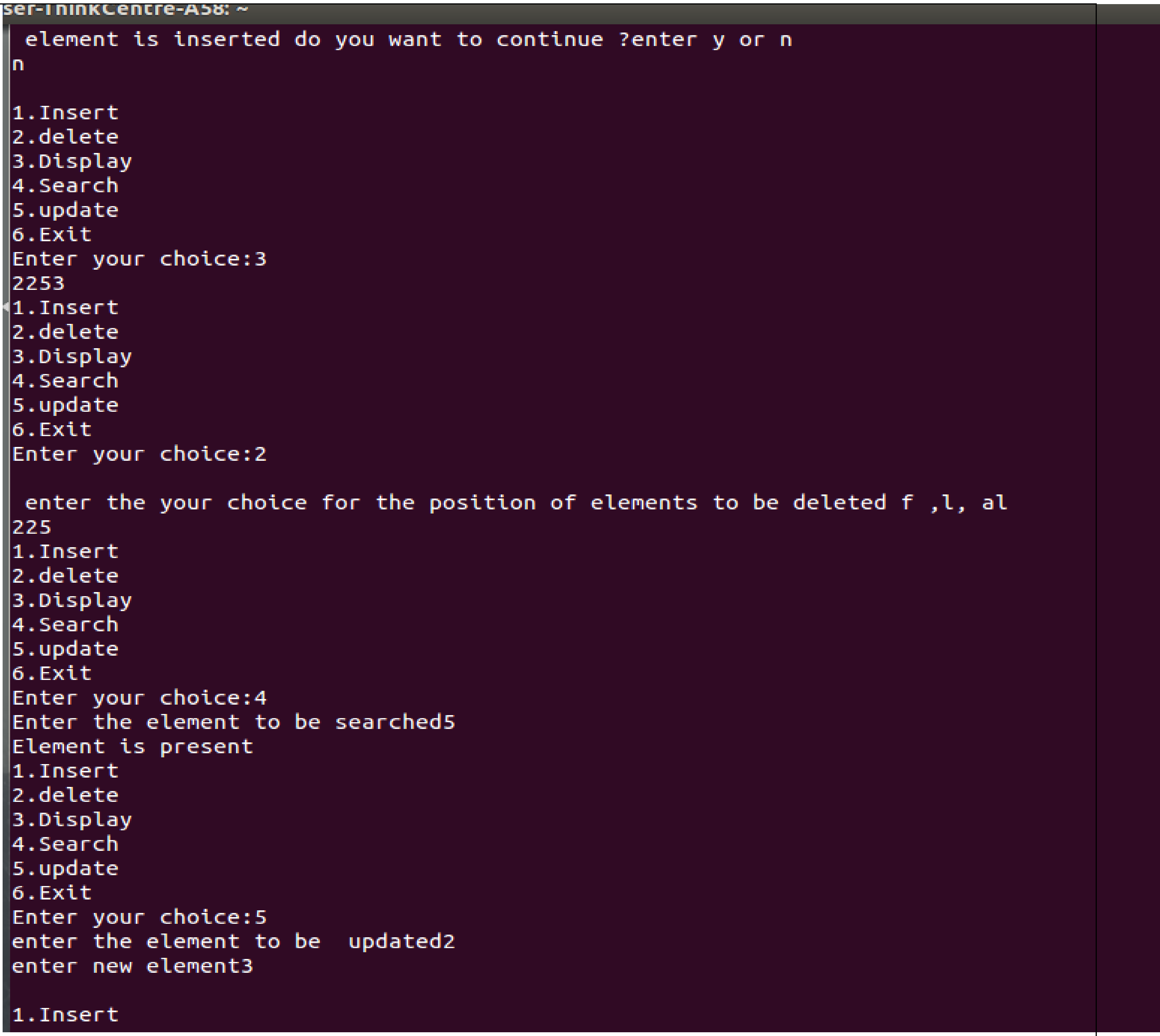
Difference between malloc() and calloc()

calloc()	malloc()
calloc() initializes the allocated memory with 0 value.	malloc() initializes the allocated memory with garbage values.
Number of arguments is 2	Number of argument is 1
Syntax : (cast_type*)calloc(blocks , size_of_block);	Syntax : (cast_type*)malloc(Size_in_bytes);

INPUT- SAMPLE EXPEXTED OUTPUT:



```
user@user-ThinkCentre-A58:~$ gcc slist.c
user@user-ThinkCentre-A58:~$ ./a.out

1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:1
Enter the element to insert2

 element is inserted
1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:1

 enter the position f ,l,a
enter ur choicef
Enter the element2

 element is inserted do you want to continue ?enter y or n
y

 enter the position f ,l,a
enter ur choicel
Enter the element3

 element is inserted do you want to continue ?enter y or n
y

 enter the position f ,l,a
enter ur choicea
Enter the element5
enter the position at which the element has to be inserted3

 element is inserted do you want to continue ?enter y or n
```



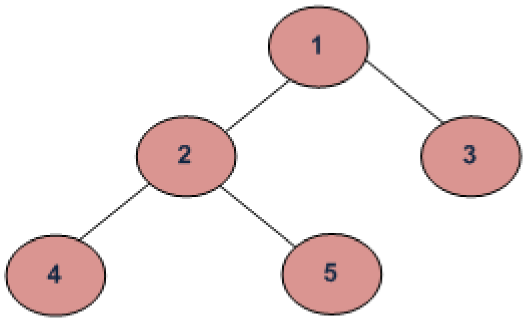
```
ser-ThinkCentre-A58: ~
 element is inserted do you want to continue ?enter y or n
n

1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:3
2253
1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:2

 enter the your choice for the position of elements to be deleted f ,l, al
225
1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:4
Enter the element to be searched5
Element is present
1.Insert
2.delete
3.Display
4.Search
5.update
6.Exit
Enter your choice:5
enter the element to be  updated2
enter new element3

1.Insert
```

CONCLUSION :-Hence program for single linked list is executed successfully and output was verified.

EXPERIMENT NO-5

AIM: To develop a program for BINARY TREE traversal.

THEORY:
Tree Traversals (Inorder, Preorder and Postorder)
Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Example Tree

Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

Inorder **Traversal :**

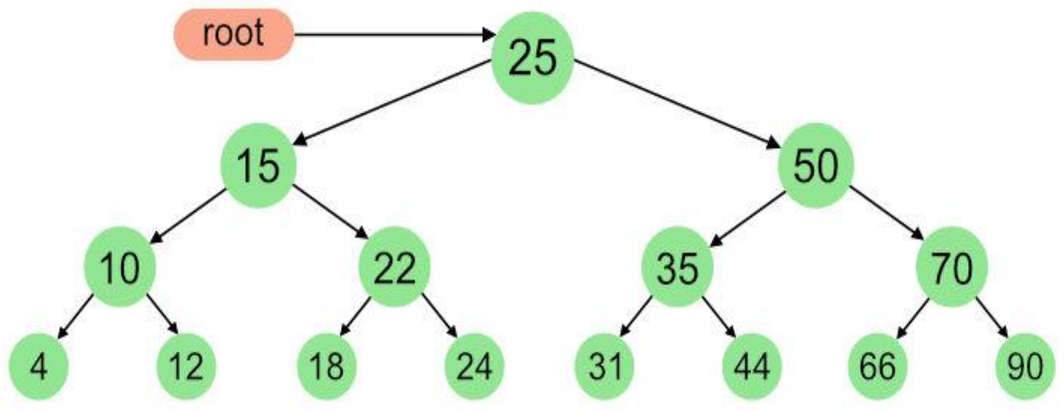
```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
```
Uses of In order
In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST

in non-increasing order, a variation of Inorder traversal where Inorder traversal s reversed can be used.
Example: Inorder traversal for the above-given figure is 4 2 5 1 3.

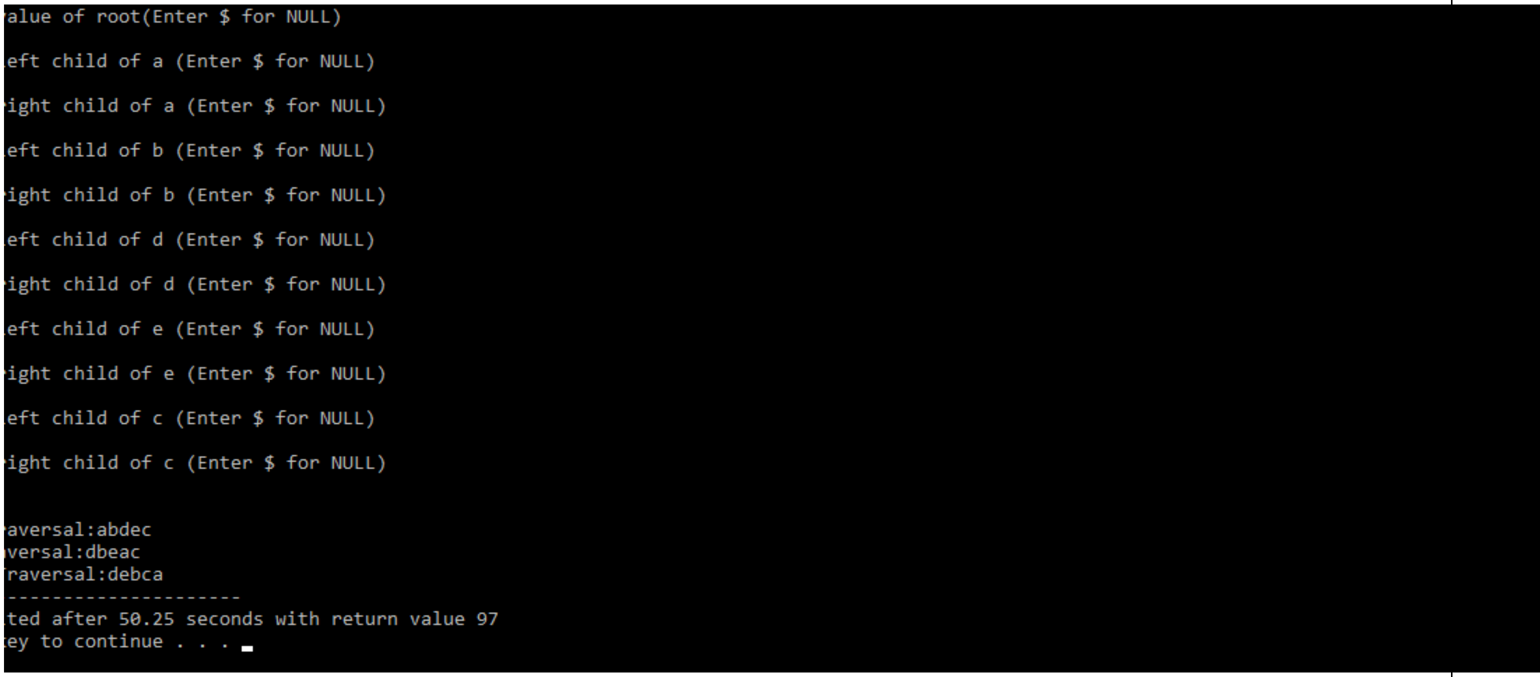
InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



INPUT- SAMPLE EXPECTED OUTPUT:



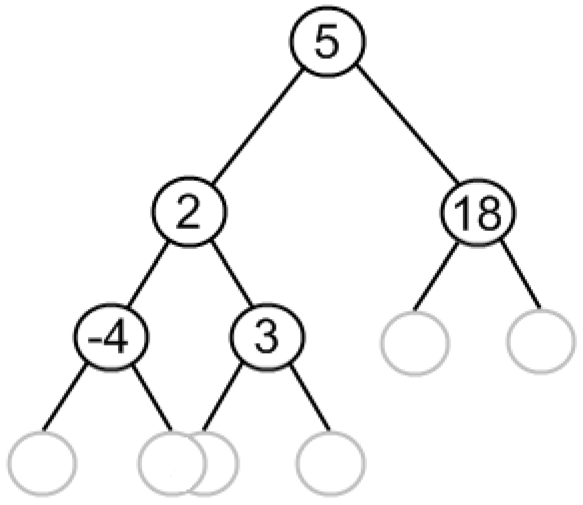
```
alue of root(Enter $ for NULL)

eft child of a (Enter $ for NULL)

ight child of a (Enter $ for NULL)

eft child of b (Enter $ for NULL)

ight child of b (Enter $ for NULL)

eft child of d (Enter $ for NULL)

ight child of d (Enter $ for NULL)

eft child of e (Enter $ for NULL)

ight child of e (Enter $ for NULL)

eft child of c (Enter $ for NULL)

ight child of c (Enter $ for NULL)


aversal:abdec
versal:dbeac
raversal:debca
--------------------
ted after 50.25 seconds with return value 97
ey to continue . . .
```

CONCLUSION:Hence,the program for binary tree traversal is executed successfully and the output was verified.

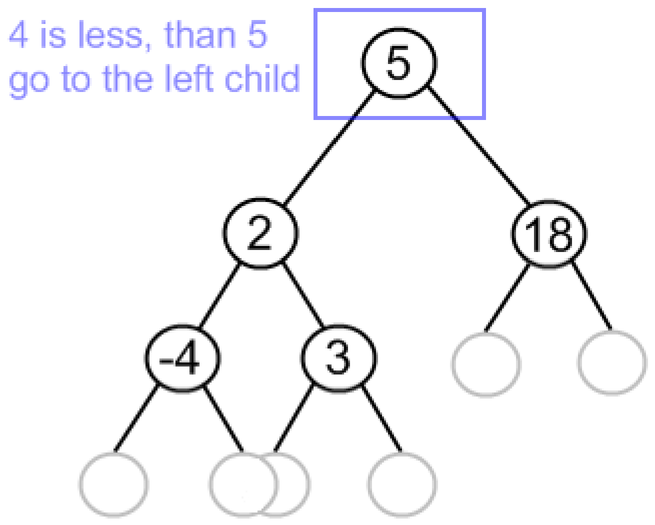
EXPERIMENT 6

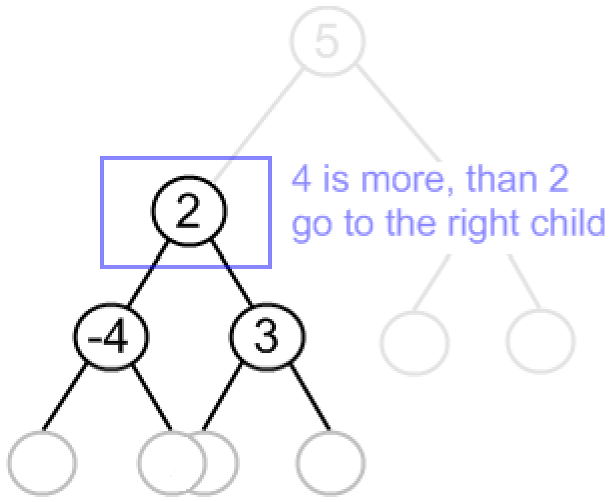
AIM: To develop a menu driven program for 'INSERTION AND SEARCHING' of binary search tree.

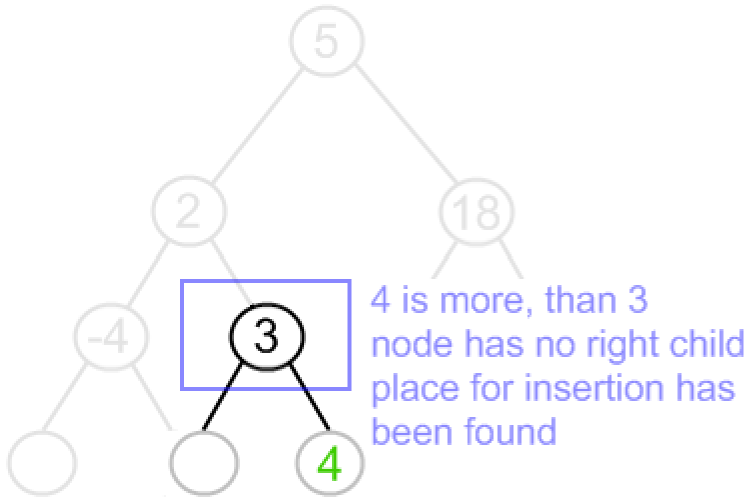
THEORY: **Binary Search Tree** is a node-based binary tree data structure which has the following properties:The left subtree of a node contains only nodes with keys lesser than the node's key.The right subtree of a node contains only nodes with keys greater than the node's key.The left and right subtree each must also be a binary search tree.Binary search tree. INSERTIONAdding a value to BST can be divided into two stages: search for a place to put a new element;insert the new element to this place.Let us see these stages in more detail. Search for a place.At this stage an algorithm should follow binary search tree property. If a new value is less, than the current node's value, go to the left subtree, else go to the right subtree. Following this simple rule, the algorithm reaches a node, which has no left or right subtree. By the moment a place for insertion is found, we can say for sure, that a new value has no duplicate in the tree. Initially, a new node has no children, so it is a leaf. Let us see it at the picture. Gray circles indicate

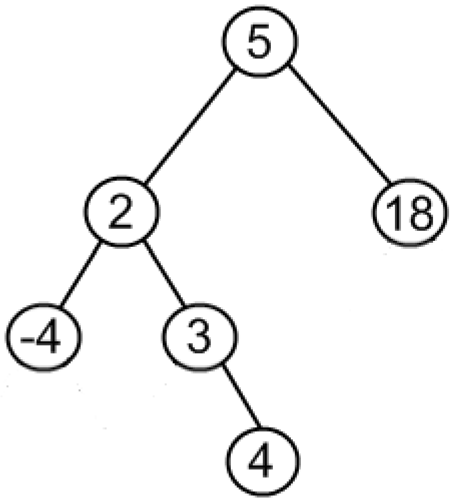


possible places for a new node. Now, let's go down to algorithm itself. Here and in almost every operation on BST recursion is utilized. Starting from the rootcheck, whether value in current node and a new value are equal. If so, duplicate is found. Otherwise,if a new value is less, than the node's value: if a current node has no left child, place for insertion has been found;otherwise, handle the left child with the same algorithm.if a new value is greater, than the node's value: if a current node has no right child, place for insertion has been found;otherwise, handle the right child with the same algorithm.Just before code snippets, let us have a look on the example, demonstrating a case of insertion in the binary search tree. ExampleInsert 4 to the tree, shown above.


4 is less, than 5
go to the left child
5
2
18
-4
3


5
2
4 is more, than 2
go to the right child
-4
3

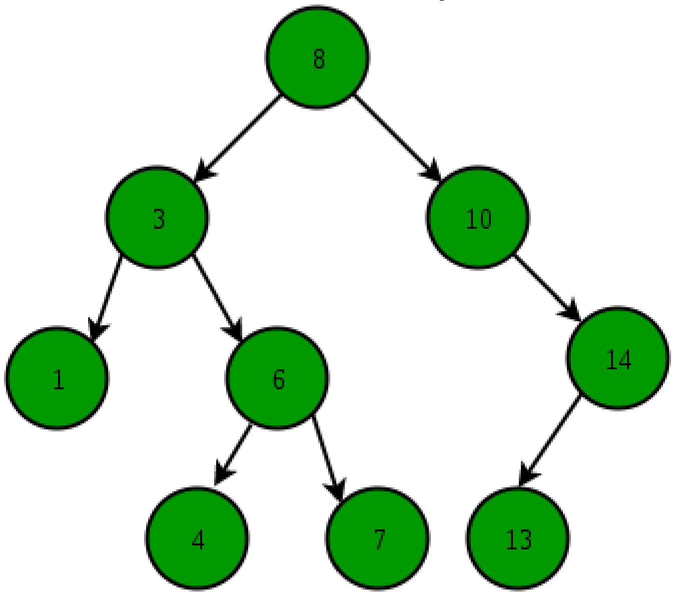




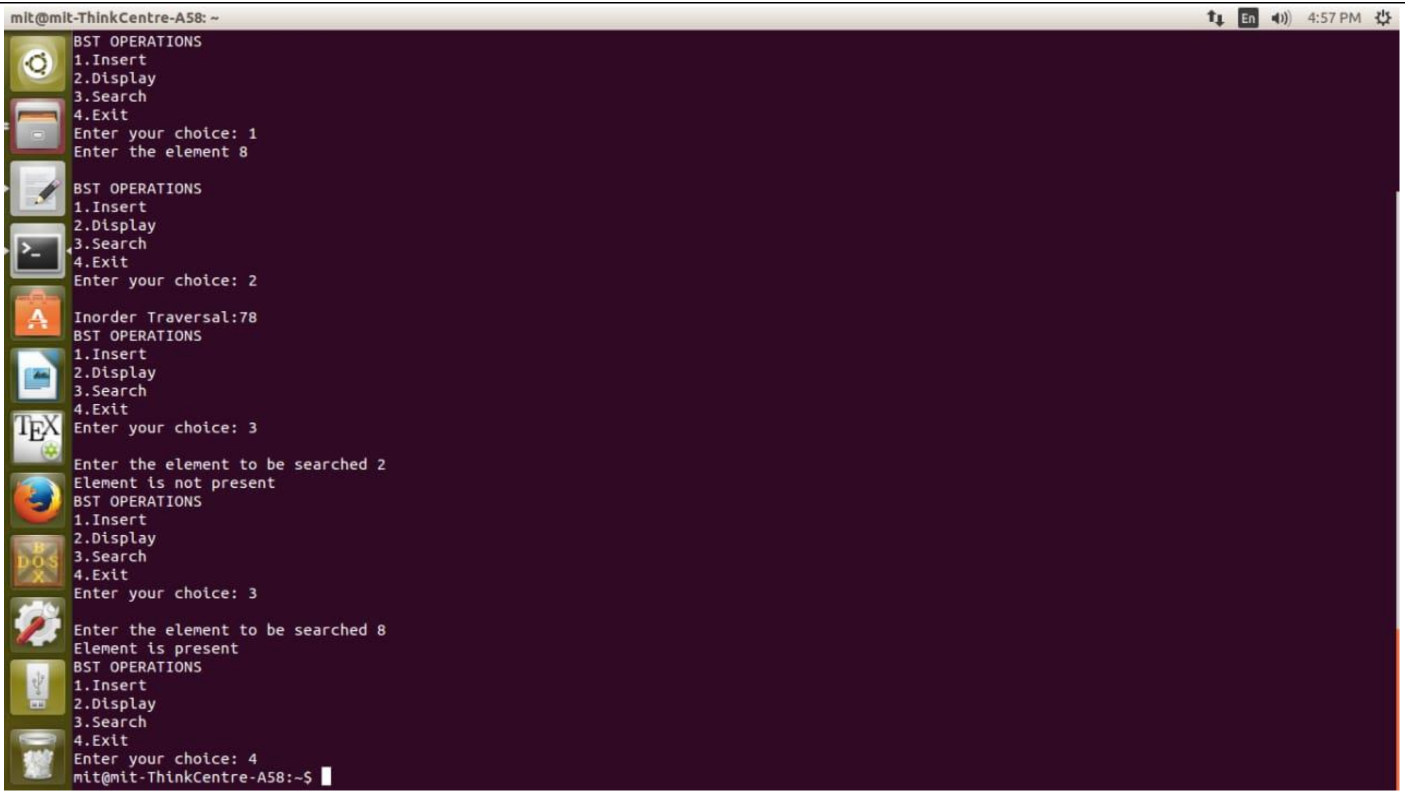
Searching a key
To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.

Illustration to search 6 in below tree:
1. Start from root.
2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.

3. If element to search is found anywhere, return true, else return false.



INPUT- SAMPLE EXPECTED OUTPUT: [redacted]



```
mit@mit-ThinkCentre-A58: ~
BST OPERATIONS
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 1
Enter the element 8

BST OPERATIONS
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 2

Inorder Traversal:78
BST OPERATIONS
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 3

Enter the element to be searched 2
Element is not present
BST OPERATIONS
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 3

Enter the element to be searched 8
Element is present
BST OPERATIONS
1.Insert
2.Display
3.Search
4.Exit
Enter your choice: 4
mit@mit-ThinkCentre-A58:~$
```

CONCLUSION: Hence, the program for binary search tree is executed successfully and the output was verified

EXPERIMENT NO.7

AIM: To develop a program for graph traversal using breadth first search .

THEORY:

A Graph G = (V, E) is a collection of sets V and E where V is a collection of vertices and E is a collection of edges.

A Graph can be of two types:
1. Directed Graph
2. Undirected Graph

In data structures, there is a popular term known as 'Traversal'. It is the process of systematically visiting or examining (may be to update the Graph nodes) each node in a tree data structure, exactly once.

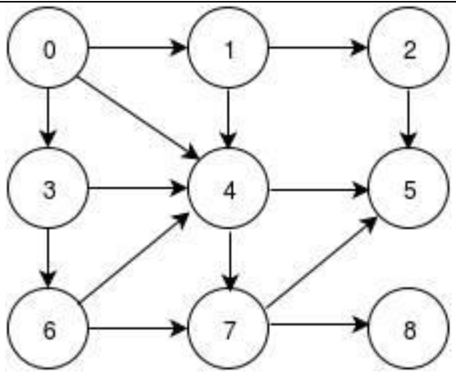
There are two most common methods to traverse a Graph:
1. Breadth First Search
2. Depth First Search

Breadth First Search technique:-

In this technique, we first visit the vertex and then visit all the vertices adjacent to the starting vertex i.e., 0. Next, we pick the adjacent vertices one after another and visit their adjacent vertices and this process goes on and on until we reach the last vertex.

Breadth First Search (BFS) Example

Consider below Graph as an example.



The vertex 0 is the starting vertex in our case. We start our traversal from the vertex 0. Then we will visit all vertices adjacent to vertex 0 i.e., 1, 4, 3. Here, we can visit these three vertices in any order. Suppose we visit the vertices in order 1,3,4.

The traversal would be: 0 1 3 4

Now, we shall visit all the vertices adjacent to 1, then all the vertices adjacent to 3 and then all the vertices adjacent to 4. So first we shall visit 2 (since it is adjacent to 1), then 6 (since it is adjacent to 3) and 5, 7 (since these are adjacent to 4).

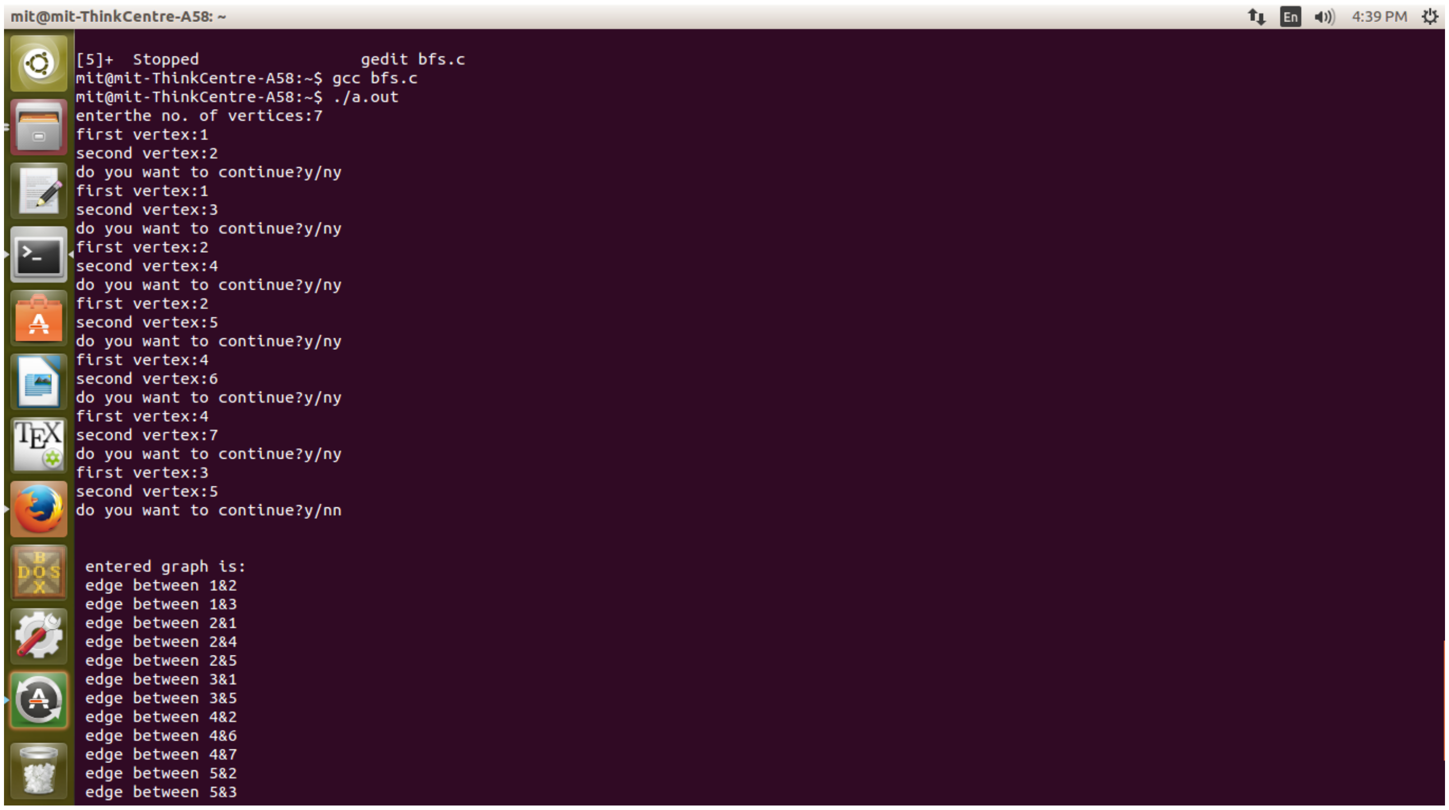
The traversal would be: 0 1 3 4 2 6 5 7

Now, we shall visit all the vertices adjacent to 2, 6, 5, and 7 one by one. We can see that vertex 5 is adjacent to vertex 2. Since, it has already been traversed upon before, we have don't need to traverse through it again and move on to the next vertex. Now, the vertices 4 and 7 are adjacent to the vertex 6. Since, they have been traversed upon before, we will not traverse on them again. Vertex 5 does not have any adjacent vertices. Vertices 5 and 8 are adjacent to vertex 7. Since, vertex 5 has been traversed upon before, we will not traverse it again. However, vertex 8 has not yet been visited. So we will traverse on vertex 8.

The traversal would be: 0 1 3 4 2 6 5 7 8

Now, we need to visit vertices adjacent to vertex 8. However, there is no vertex adjacent to vertex 8 and hence we will have to stop the traversal here.

INPUT- SAMPLE EXPECTED OUTPUT:



```
[5]+  Stopped                 gedit bfs.c
mit@mit-ThinkCentre-A58:~$ gcc bfs.c
mit@mit-ThinkCentre-A58:~$ ./a.out
enterthe no. of vertices:7
first vertex:1
second vertex:2
do you want to continue?y/ny
first vertex:1
second vertex:3
do you want to continue?y/ny
first vertex:2
second vertex:4
do you want to continue?y/ny
first vertex:2
second vertex:5
do you want to continue?y/ny
first vertex:4
second vertex:6
do you want to continue?y/ny
first vertex:4
second vertex:7
do you want to continue?y/ny
first vertex:3
second vertex:5
do you want to continue?y/nn


 entered graph is:
 edge between 1&2
 edge between 1&3
 edge between 2&1
 edge between 2&4
 edge between 2&5
 edge between 3&1
 edge between 3&5
 edge between 4&2
 edge between 4&6
 edge between 4&7
 edge between 5&2
 edge between 5&3
```

CONCLUSION: HENCE PROGRAM FOR GRAPH TRAVERSAL USING BFS WAS EXECUTED SUCCESFULLY AND OUTPUT WAS VERIFIED.

	G.S. MANDAL'S **MAHARASHTRA INSTITUTE OF TECHNOLOGY, AURABGABAD**	
	LAB WORK INSTRUCTION SHEET	
	DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING	
CLASS: S.Y. B. TECH PART: 1 (2019-20)	**LAB: 512-B**	**SUBJECT: CSE205-Data Structures**

EXPERIMENT NO.8

AIM: To develop a program to sort the numbers in ascending order using Bubble Sort .

THEORY:
Like other primitive Bubble sort is one of the simplest sorting algorithms. The two adjacent elements of an array are checked and swapped if they are in wrong order and this process is repeated until we get a sorted array. The steps of performing a bubble sort are: Compare the first and the second element of the array and swap them if they are in wrong order.Compare the second and the third element of the array and swap them if they are in wrong order. Proceed till the last element of the array in a similar fashion.Repeat all of the above steps until the array is sorted.This will be more clear by the following visualizations.
Initial array

16	19	11	15

First iteration

16	19	11	15

16	19	11	15

SWAP

16	11	19	15

16	11	19	15

SWAP

16	11	15	19

Second iteration

16	11	15	19

SWAP

11	16	15	19

11	16	15	19

SWAP

11	15	16	19

11	15	16	19

Third iteration

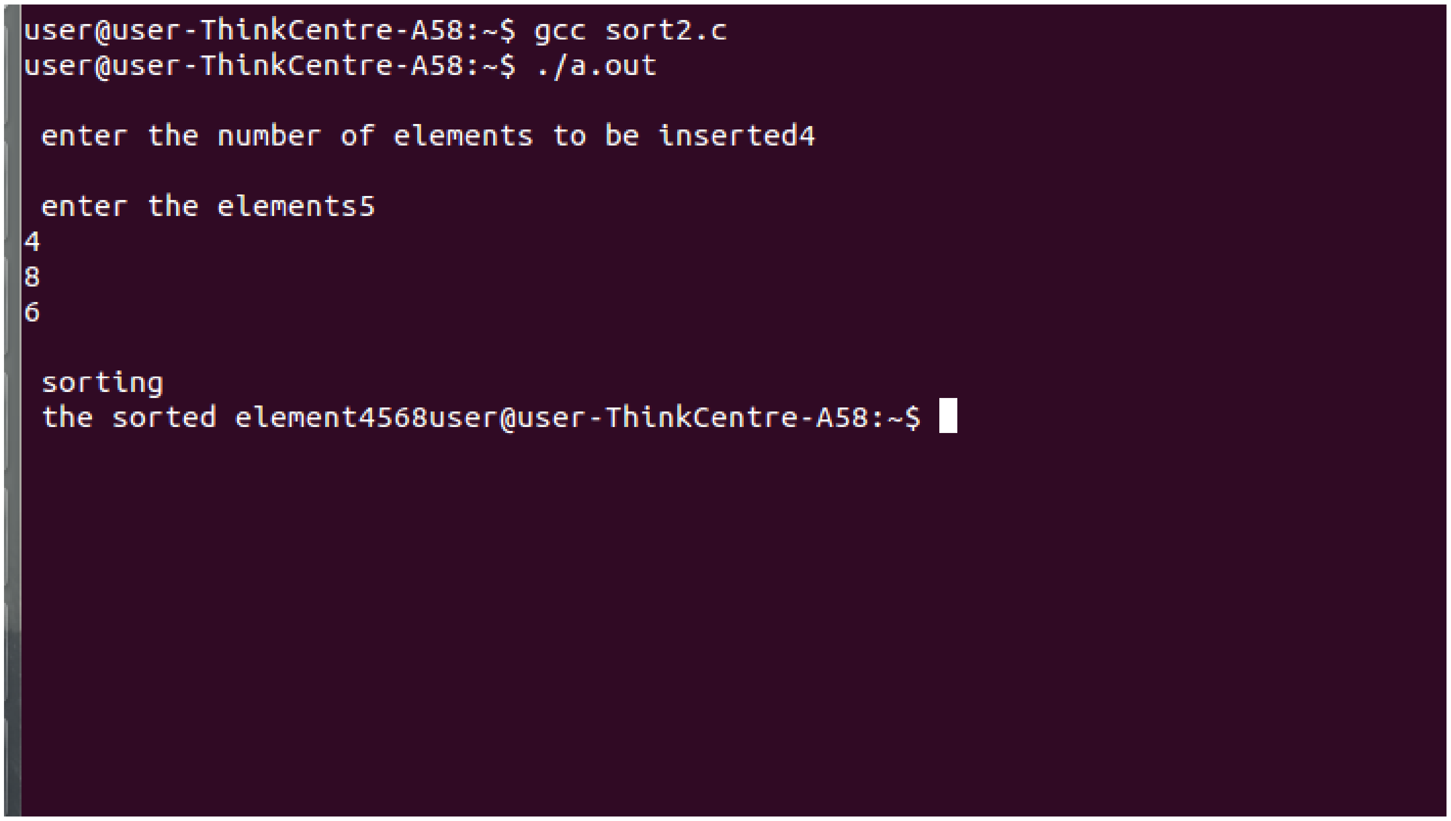
11	15	16	19

11	15	16	19

11	15	16	19

No swap → Sorted → break the loop

INPUT- SAMPLE EXPECTED OUTPUT:



```
user@user-ThinkCentre-A58:~$ gcc sort2.c
user@user-ThinkCentre-A58:~$ ./a.out

 enter the number of elements to be inserted4

 enter the elements5
4
8
6

 sorting
 the sorted element4568user@user-ThinkCentre-A58:~$
```

CONCLUSION:-Thus the program for Bubble Sort was executed and the output was verified.

EXPERIMENT NO.9

AIM: To develop a program for quick sort using array .

THEORY:
Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value. Quick sort is the fastest internal sorting algorithm with the time complexity O (n log n). The basic algorithm to sort an array a[ ] of n elements can be described recursively as follows:

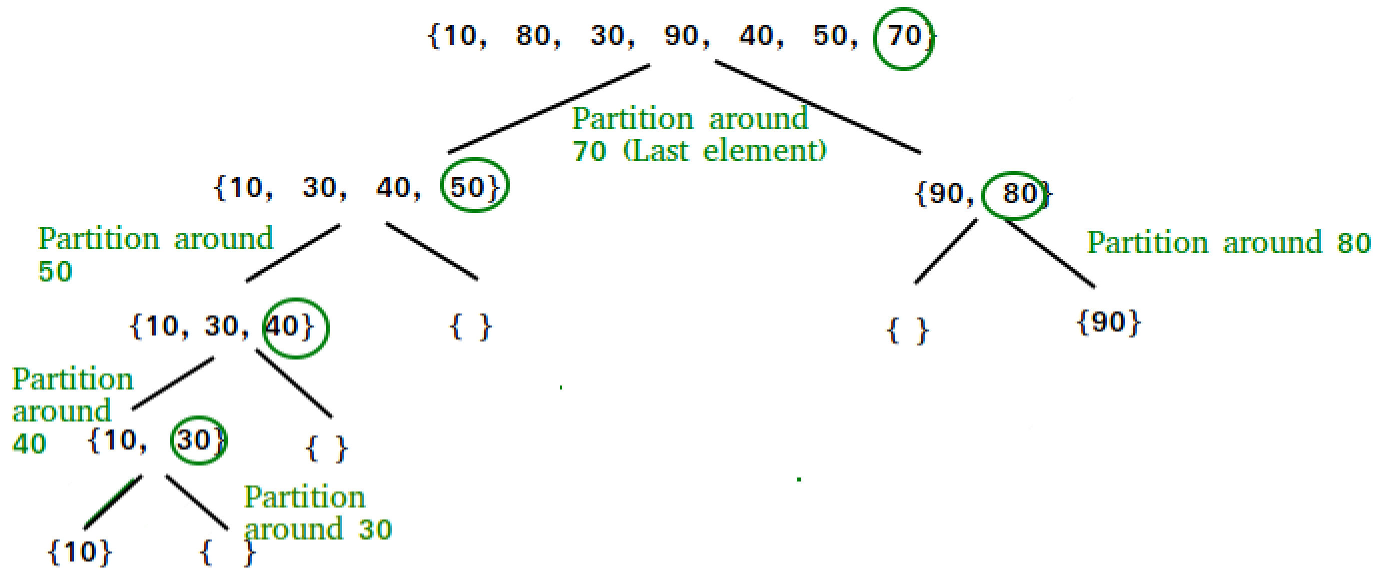
Quick Sort Algorithm

1. If n < = 1, then return.

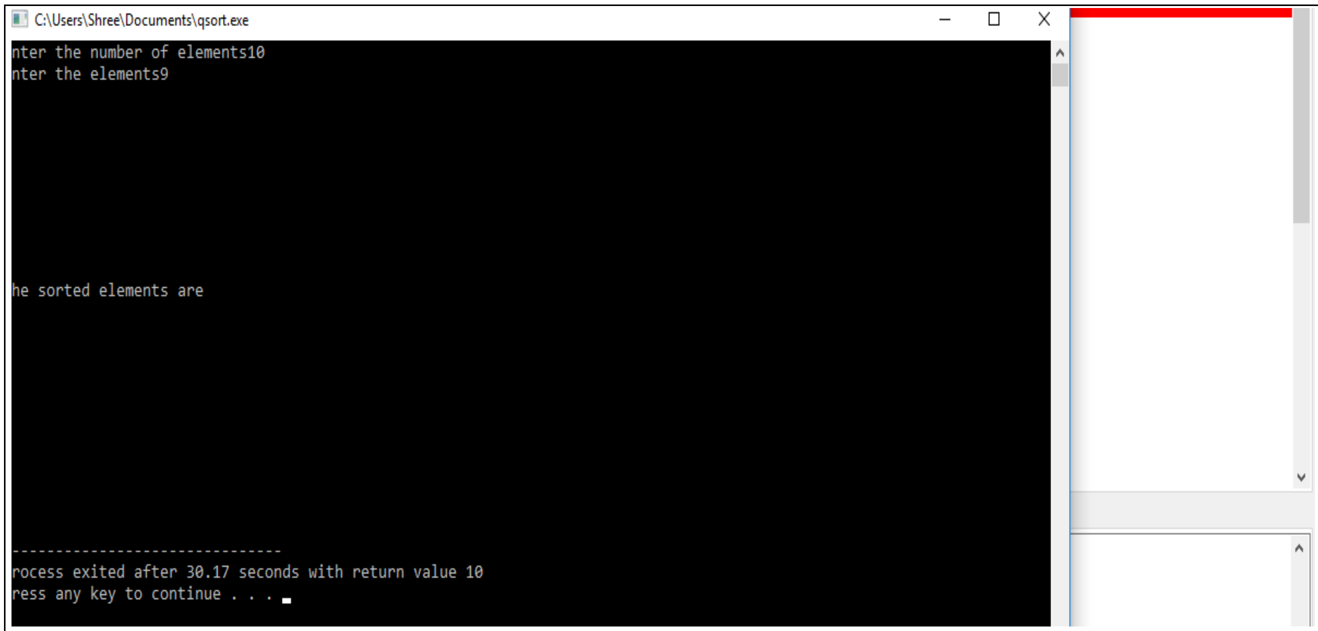
2. Pick any element V in a[]. This is called the pivot.

3. Rearrange elements of the array by moving all elements xi > V right of V and all elements xi < = V left of V. If the place of the V after re-arrangement is j, all elements with value less than V, appear in a[0], a[1] . . . . a[j – 1] and all those with value greater than V appear in a[j + 1] . . . . a[n – 1].

4. Apply quick sort recursively to a[0] . . . . a[j – 1] and to a[j + 1] . . . . a[n – 1].



INPUT- SAMPLE EXPECTED OUTPUT:


C:\Users\Shree\Documents\qsort.exe

```
nter the number of elements10
nter the elements9



he sorted elements are




------------------------------
rocess exited after 30.17 seconds with return value 10
ress any key to continue . . .
```

CONCLUISON: HENCE PROGRAM FOR QUICK SORT WAS EXECUTED SUCCESFULLY AND OUTPUT WAS VERIFIED.

EXPERIMENT NO.10

AIM: To develop a menu driven program for 'LINEAR AND BINARY SEARCH ' using array.

THEORY:

Linear search and binary search are the two methods which are used in arrays for **searching** the elements. Searching is a process of finding an element within the list of elements stored in any order or randomly.

Definition of Binary Search

Binary search is an extremely efficient algorithm. This search technique consume less time in searching the given item in minimum possible comparisons. To do the binary search, first, we have to sort the array elements.

ALGORITHM:First, find the middle element of the array.The middle element of the array is compared to the element to be searched.If the element is the required element, then the search is successful.When the element is less than the desired item, then search only the first half of the array.If it is greater than the desired element, then search in the second half of the array.

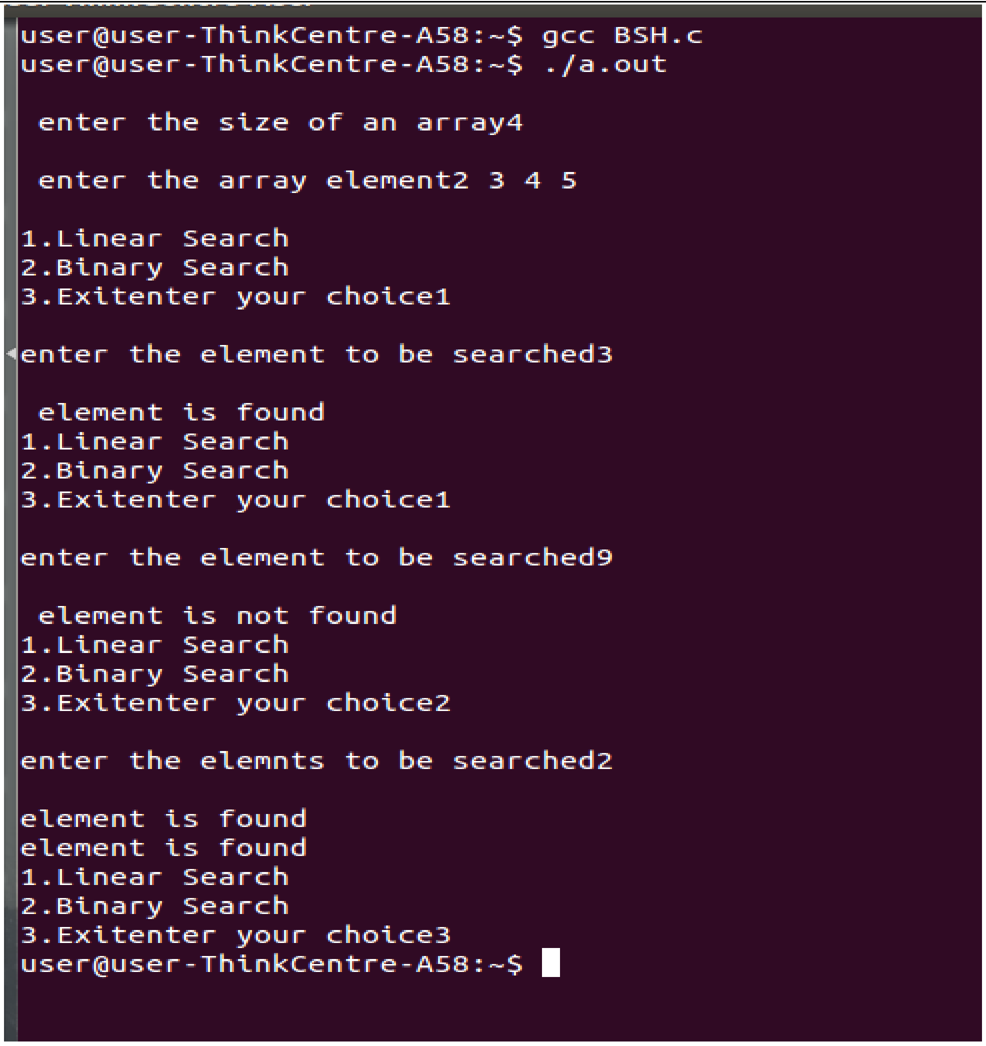
Repeat the same steps until an element is found or exhausts in the search area. In this algorithm, every time search area is reducing. Therefore the number of comparisons is at most log (N+1). As a result, it is an efficient algorithm when compared to linear search, but the array has to be sorted before doing the binary search. Both linear and binary search algorithms can be useful depending on the application. When an array is the data structure and elements are arranged in sorted order, then binary search is preferred for **quick** searching. If linked list is the data structure regardless how the elements are arranged, linear search is adopted due to unavailability of direct implementation of binary search algorithm.

Both linear and binary search algorithms can be useful depending on the application. When an array is the data structure and elements are arranged in sorted order, then binary search is preferred for **quick** searching. If linked list is the data structure regardless how the elements are arranged, linear search is adopted due to unavailability of direct implementation of binary search algorithm.

The typical Binary search algorithm cannot be employed to linked list because linked list is dynamic in nature and it is not known where the middle element is actually assigned. Hence, there is a requirement to design the variation of the binary search algorithm that can work on linked list too because the binary search is faster in execution than a linear search.

INPUT- SAMPLE EXPECTED OUTPUT:

CLASS: S.Y. B. TECH PART: 1 (2019-20)	**LAB: 512-B**	**SUBJECT: CSE205-Data Structures**



```
user@user-ThinkCentre-A58:~$ gcc BSH.c
user@user-ThinkCentre-A58:~$ ./a.out

 enter the size of an array4

 enter the array element2 3 4 5

1.Linear Search
2.Binary Search
3.Exitenter your choice1

enter the element to be searched3

 element is found
1.Linear Search
2.Binary Search
3.Exitenter your choice1

enter the element to be searched9

 element is not found
1.Linear Search
2.Binary Search
3.Exitenter your choice2

enter the elemnts to be searched2

element is found
element is found
1.Linear Search
2.Binary Search
3.Exitenter your choice3
user@user-ThinkCentre-A58:~$
```

CONCLUSION: Hence, the program for linear and binary search was executed and the output was verified.