

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**III SEMESTER - R 2017**

**CS8381– DATA STRUCTURES LABORATORY**

**LABORATORY MANUAL**

Name : \_\_\_\_\_

Register No : \_\_\_\_\_

Section : \_\_\_\_\_

### VISION

is committed to provide highly disciplined, conscientious and enterprising professionals conforming to global standards through value based quality education and training.

### MISSION

- To provide competent technical manpower capable of meeting requirements of the industry
- To contribute to the promotion of Academic excellence in pursuit of Technical Education at different levels
- To train the students to sell his brawn and brain to the highest bidder but to never put a price tag on heart and soul

[www.FirstRanker.com](http://www.FirstRanker.com)

## **PROGRAM EDUCATIONAL OBJECTIVES**

### **1. FUNDAMENTALS**

To impart students with fundamental knowledge in Mathematics, Science and fundamentals of engineering that will mould them to be successful professionals

### **2. CORE COMPETENCE**

To provide students with sound knowledge in engineering and experimental skills to identify complex software problems in industry and to develop practical solutions for them

### **3. BREADTH**

To provide relevant training and experience to bridge the gap between theory and practice this enables them to find solutions for real time problems in industry and organization, and to design products requiring interdisciplinary skills

### **4. PROFESSIONALISM SKILLS**

To bestow students with adequate training and provide opportunities to work as team that will build up their communication skills, individual leadership and supportive qualities, and to enable them to adapt and work in ever changing technologies

### **5. LIFELONG LEARNING**

To develop the ability of students to establish themselves as professionals in Computer Science and Engineering and to create awareness about the need for lifelong learning and pursuing advanced degrees

## PROGRAM OUTCOMES

- a) To apply the basic knowledge of Mathematics, Science and engineering fundamentals in Computer Science and Engineering field
- b) To design and conduct experiments as well as to analyze and interpret data and apply the same in the career
- c) To design and develop innovative and creative software applications
- d) To understand a complex real world problem and develop an efficient practical solution
- e) To create, select and apply appropriate techniques, resources, modern engineering and IT tools
- f) To understand their roles as a professionals and give the best to the society
- g) To develop a system that will meet expected needs within realistic constraints such as economical, environmental, social, political, ethical, safe and sustainable
- h) To communicate effectively and make others understand exactly what they are trying to convey in both verbal and written forms
- i) To work in a team as team member or a leader and make unique contributions and work with coordination
- j) To exhibit confidence in self-education and ability for lifelong learning
- k) To develop and manage projects in multidisciplinary environments

**CS8381 – DATA STRUCTURES LABORATORY****SYLLABUS****COURSE OBJECTIVES**

- To implement linear and non-linear data structures
- To understand the different operations of search trees
- To implement graph traversal algorithms
- To get familiarized to sorting and searching algorithms

**LIST OF EXPERIMENTS:**

1. Array implementation of Stack and Queue ADTs
2. Array implementation of List ADT
3. Linked list implementation of Stack, Queue and List ADTs
4. Applications of List, Stack and Queue ADTs
5. Implementation of Binary Trees and operations of Binary Trees
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues
9. Graph representation and Traversal algorithms
10. Applications of Graphs
11. Implementation of searching and sorting algorithms
12. Hashing – any two collision techniques

**TOTAL: 60 PERIODS****COURSE OUTCOMES****Upon completion of the course, students will be able to:**

- Write functions to implement linear and non-linear data structure operations
- Suggest appropriate linear / non-linear data structure operations for solving a given problem
- Appropriately use the linear / non-linear data structure operations for a given problem
- Apply appropriate hash functions that result in a collision free scenario for data storage and retrieval

**CS8381 – DATA STRUCTURES LABORATORY  
CONTENTS**

<b>Sl.No.</b>	<b>Name of the Experiment</b>	<b>Page No.</b>
1.a)	Array implementation of Stack ADT	
1.b)	Array implementation of Queue ADT	
2.	Array implementation of List ADT	
3. a)	Linked list implementation of Stack ADT	
3. b)	Linked list implementation of Queue ADT	
3. c)	Linked list implementation of List ADT	
4. a)	Application of Stack ADT	
4. b)	Application of Queue ADT	
5.	a. Graph Representation b. Graph Traversal – DFS and BFS	
6.	Implementation of Searching algorithms - Linear Search and Binary Search	
7.	Implementation of Sorting algorithms - Bubble Sort, Shell Sort, Radix Sort	
8.	Implementation of Heaps	
9.	Implementation of Binary Trees and operations of Binary trees	
10.	Implementation of Binary Search Tree	
11	Implementation of Hashing Technique	

**Ex. No: 1a****IMPLEMENTATION OF STACK ADT****Date:****Aim:**

To write a C program to implement Stack ADT by using arrays

**Algorithm:**

1. Create a Stack[ ] with MAX size as your wish.
2. Write function for all the basic operations of stack - PUSH(), POP() and DISPLAY().
3. By using Switch case, select push() operation to insert element in the stack.
  - Step 1: Check whether stack is FULL. (top == SIZE-1)
  - Step 2: If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function.
  - Step 3: If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value).
4. Similarly, By using Switch case, select pop() operation to delete element from the stack.
  - Step 1: Check whether stack is EMPTY. (top == -1)
  - Step 2: If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function.
  - Step 3: If it is NOT EMPTY, then delete stack[top] and decrement top value by one (top--).
5. Similarly, By using Switch case, select display() operation to display all element from the stack.
  - Step 1: Check whether stack is EMPTY. (top == -1)
  - Step 2: If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function.
  - Step 3: If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--).
  - Step 3: Repeat above step until i value becomes '0'.
6. Close the program

**Sample output :**

Enter the size of STACK[MAX=100]:10

STACK OPERATIONS USING ARRAY

-----

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:1

Enter a value to be pushed:12

Enter the Choice:1

Enter a value to be pushed:24

Enter the Choice:1  
Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98

24

12

Press Next Choice

Enter the Choice:2

The popped elements is 98

Enter the Choice:3

The elements in STACK

24

12

Press Next Choice

Enter the Choice:4

EXIT POINT

#### **Result:**

Thus the C program to implement Stack ADT by using array is executed successfully and the output is verified.

#### **Program Outcome:**

Thus the student can know to implement Linear Data Structure – Stack by using array

#### **Applications:**

1. Expression evaluation
2. Function call and return process.



**Ex. No: 1b****IMPLEMENTATION OF QUEUE ADT****Date:****Aim:**

To write a C program to implement Queue ADT by using arrays

**Algorithm:**

1. Create a Queue[ ] with MAX size as your wish.
2. Write function for all the basic operations of stack - Enqueue(), Dequeue() and Display().
3. By using Switch case, select Enqueue() operation to insert element in to the rear/back end of the queue.
  - Check whether queue is FULL. (rear == SIZE-1)
  - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
  - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value
4. Similarly, by using Switch case, select Dequeue() function is used to remove the element from the front end of the queue.
  - Check whether queue is EMPTY. (front == rear)
  - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.
  - Step 3: If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).
5. Similarly, by using Switch case, select display() operation to display all element of the queue.
  - Step 1: Check whether queue is EMPTY. (front == rear)
  - Step 2: If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.
  - Step 3: If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.
  - Step 3: Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value is equal to rear (i <= rear)
6. Close the program

**Output:**

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1

Inset the element in queue : 10

- 1.Insert element to queue

2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 1

Inset the element in queue : 15

1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 1

Inset the element in queue : 20

1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 1

Inset the element in queue : 30

1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 2

Element deleted from queue is : 10

1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 3

Queue is :

15 20 30

1.Insert element to queue  
2.Delete element from queue  
3.Display all elements of queue  
4.Quit

Enter your choice : 4

**Result:**

Thus the C program to implement Stack ADT by using array is executed successfully and the output is verified.

**Program Outcome:**

Thus the student can know to implement Linear Data Structure – Queue by using array

**Applications:**

1. Queue is used by Operating systems for job scheduling.
2. Queue is used in networking to handle congestion.

**Viva Voce:**

1. What is linear and Non-linear data structure?
2. What is array?
3. Define ADT.
4. What is Stack and list the operations of stack?
5. What is Queue and list the operations of queue?
6. How the insertion and deletion takes place in stack?
7. How the insertion and deletion takes place in queue?
8. What is push in stack?
9. What is pop in stack?
10. What is enqueue?
11. What is dequeue?

**EX NO: 2****IMPLEMENTATION OF LIST ADT****DATE****Aim:**

To write a C program to implement List ADT by using arrays

**Algorithm:**

1. Create an array[ ] with MAX size as your wish.
2. Write function for all the basic operations of list - create(), insert(), deletion(), search(), display().
3. By using Switch case, select create() operation to create the list.
  - List Will be in the form:  $A_1, A_2, A_3, \dots A_n$
  - First Element of the List:  $A_1$
  - Last Element of the List:  $A_n$
  - $i$ th Element of the List:  $A_i$ ,
  - Position of Element  $A_i$  :  $i$  , Position ranges from 0 to N
  - Size of the List: N
  - Empty List: If Size of the list is Zero (i.e  $N=0$ ), it is Empty List.
  - Precedes and Succeeds: For any list except empty list, we say that  $A_{i+1}$  follows (or succeeds)  $A_i$  ( $i < N$ ) and that  $A_{i-1}$  precedes  $A_i$  ( $i > 1$ )
4. Similarly, by using Switch case, select insert() operation to insert element in the list.
  - Insert x at position p in list L
  - If  $p = \text{END}(L)$ , insert x at the end
  - If L does not have position p, result is undefined
5. Similarly, by using Switch case, select delete() function is used to remove the element from the list.
  - delete element at position p in L
  - undefined if  $p = \text{END}(L)$  or does not exist
6. Similarly, by using Switch case, select search() function is used to retrieve the required element from the list if it is available.
  - returns element at position p on L
  - undefined if p does not exist or  $p = \text{END}(L)$
7. Similarly, by using Switch case, select display() operation to display all element of the list
  - print the elements of L in order of occurrence
8. Close the program

**Output:**

Main Menu

1. Create
2. Delete
3. Search

```
4. Insert
5.Display
6.Exit
Enter your Choice: 3
Enter the number of nodes 3
Enter the element: 67
Enter the element: 69
Enter the element: 71
Do You Want to continue: y
Main Menu
1. Create
2. Delete
3. Search
4. Insert
5.Display
6.Exit
Enter your Choice: 4
Enter the position u need to insert: 2
Enter the element to insert: 56
The list after insertion:
    The elements of the list ADT are:
67
69
56
71
Do You Want to continue: y
Main Menu
1.Create
2. Delete
3. Search
4. Insert
5.Display
6.Exit
Enter your Choice: 2
Enter the position u want to delete: 2
The elements after deletion: 67 69 71
Do You Want to continue: y
Main Menu
1.Create
2. Delete
3. Search
4. Insert
5.Display
6.Exit
Enter your Choice: 3
Enter the element to be searched: 69
Value is in 1 position
Do You Want to continue: y
Main Menu
```

- 1.Create
2. Delete
3. Search
4. Insert
- 5.Display
- 6.Exit

Enter your Choice: 5

The elements of the list ADT are:

67

69

71

Do You Want to continue: N

**Result:**

Thus the C program to implement List ADT by using array is executed successfully and the output is verified.

**Program Outcome:**

Thus the student can know to implement Linear Data Structure – List by using array

**Applications:**

1. It is very much useful in sorting algorithms.  
Example: Selection sort

**Viva Voce:**

1. What is array?
2. Define ADT.
3. What is List?
4. Differentiate List and Array.
5. What are the different operations involved in List?
6. How the search operation works in list?
7. How will you find the list is full?
8. How will you find the list is empty?
9. Is the list belongs to linear data structure? Justify
10. Differentiate List and Linked list.
- 11.

**Ex. No. : 3a LINKED LIST IMPLEMENTATION OF STACK ADT****Date:****Aim:**

To write a C program to implement Stack ADT by using linked list

**Algorithm:**

1. Include all the header files which are used in the program. And declare all the user defined functions.
2. Define a 'Node' structure with two members data and next.
3. Define a Node pointer 'top' and set it to NULL.
4. Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.
5. push(value) - Inserting an element into the Stack
  - Create a newNode with given value.
  - Check whether stack is Empty (top == NULL)
  - If it is Empty, then set newNode → next = NULL.
  - If it is Not Empty, then set newNode → next = top.
  - Finally, set top = newNode.
6. pop() - Deleting an Element from a Stack
  1. Check whether stack is Empty (top == NULL).
  2. If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
  3. If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
  4. Then set 'top = top → next'.
  5. Finally, delete 'temp' (free(temp)).
7. display() - Displaying stack of elements
  1. Check whether stack is Empty (top == NULL).
  2. If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
  3. If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
  4. Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack (temp → next != NULL).
  5. Finally! Display 'temp → data ---> NULL'.

**Output**

\*\*\*\*\* MENU \*\*\*\*\*

1. Push
2. Pop
3. Display
4. Exit

Enter your choice:1

Enter the value to be insert: 3

Insertion Success

\*\*\*\*\* MENU \*\*\*\*\*

1. Push
2. Pop
3. Display
4. Exit

Enter your choice:1

Enter the value to be insert: 5

Insertion Success

\*\*\*\*\* MENU \*\*\*\*\*

1. Push
2. Pop
3. Display
4. Exit

Enter your choice:2

Deleted Element: 5

\*\*\*\*\* MENU \*\*\*\*\*

1. Push
2. Pop
3. Display
4. Exit

Enter your choice:3

3->->->NULL

\*\*\*\*\* MENU \*\*\*\*\*

1. Push
2. Pop
3. Display
4. Exit

Enter your choice:4

...Program finished with exit code 0

#### Result:

Thus the C program to implement Stack ADT by using linked list is executed successfully and the output is verified.

#### Program Outcome:

Thus the student can know to implement Linear Data Structure – Stack by using linked list

#### Applications:

1. It is very much useful in memory management
- 2.



**Ex. No. : 3b****Date:****LINKED LIST IMPLEMENTATION OF QUEUE ADT****Aim:**

To write a C program to implement Queue ADT by using linked list

**Algorithm:**

1. Include all the header files which are used in the program. And declare all the user defined functions.
2. Define a 'Node' structure with two members data and next.
3. Define two Node pointers 'front' and 'rear' and set both to NULL.
4. Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.
5. enqueue(value) - Inserting an element into the queue
  - Create a newNode with given value and set 'newNode → next' to NULL.
  - Check whether queue is Empty (rear == NULL)
  - If it is Empty then, set front = newNode and rear = newNode.
  - If it is Not Empty then, set rear → next = newNode and rear = newNode.
6. dequeue() - Deleting an Element from queue
  - Check whether queue is Empty (front == NULL).
  - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
  - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
  - Then set 'front = front → next' and delete 'temp' (free(temp)).
7. display() - Displaying queue of elements
  - Check whether queue is Empty (front == NULL).
  - If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
  - If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
  - Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
  - Finally! Display 'temp → data ---> NULL'.

**Sample Output**

:: Queue Implementation using Linked List ::

\*\*\*\*\* MENU \*\*\*\*\*

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1

Enter the value to be insert: 5

Insertion is Success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1

Enter the value to be insert: 7

Insertion is Success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 2

Deleted element : 5

\*\*\*\*\* MENU \*\*\*\*\*

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 3

7→>>> NULL

#### Result:

Thus the C program to implement Queue ADT by using linked list is executed successfully and the output is verified.

#### Program Outcome:

Thus the student can know to implement Linear Data Structure – Queue by using linked list

#### Applications:

1. CPU scheduling algorithms implemented by using queue
2. In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
- 3.

**Ex. No. : 3c**
**Date:**
**LINKED LIST IMPLEMENTATION OF LIST ADT**
**Aim:**

To write a C program to implement List ADT by using linked list

**Algorithm:**

1. Create the header file Llist.h header file and we are include there header file into the main function program by through `#include<Llist.h>`
2. Write function for all the basic operations of list - create(), insert(), deletion(), search(), display().
3. By using Switch case, select create() operation to create the list.
  - List Will be in the form:  $A_1, A_2, A_3, \dots A_n$
  - First Element of the List:  $A_1$
  - Last Element of the List:  $A_n$
  - ith Element of the List:  $A_i$ ,
  - Position of Element  $A_i : i$ , Position ranges from 0 to N
  - Size of the List: N
  - Empty List: If Size of the list is Zero (i.e  $N=0$ ), it is Empty List.
  - Precedes and Succeeds: For any list except empty list, we say that  $A_{i+1}$  follows (or succeeds)  $A_i$  ( $i < N$ ) and that  $A_{i-1}$  precedes  $A_i$  ( $i > 1$ )
4. Similarly, by using Switch case, select insert() operation to insert element in the list.
  - Insert x at position p in list L
  - If  $p = \text{END}(L)$ , insert x at the end
  - If L does not have position p, result is undefined
5. Similarly, by using Switch case, select delete() function is used to remove the element from the list.
  - delete element at position p in L
  - undefined if  $p = \text{END}(L)$  or does not exist
6. Similarly, by using Switch case, select search() function is used to retrieve the required element from the list if it is available.
  - returns element at position p on L
  - undefined if p does not exist or  $p = \text{END}(L)$
7. Similarly, by using Switch case, select display() operation to display all element of the list
  - print the elements of L in order of occurrence
8. Close the program

**Sample Output:**

List Adt Using Linked List

1.Create

2.Insert

```
3.Delete
4.MakeEmpty
5.Find
6.IsEmpty
7.Display
8.Deletelist
9.Exit
Enter ur Option: 1
List is Created.
Enter ur Option: 2
Enter the value: 100
Enter the Option: 2
Enter the value: 200
Enter the Option: 2
Enter the value: 300
Enter the Option: 2
Enter the value: 400
Enter the Option: 2
Enter the value: 500
Enter the Option: 7
100
200
300
400
500
Enter the Option: 3
Enter the value to delete: 200
Enter the Option: 7
100
300
400
500
Enter the Option: 5
Enter the value to Find: 400
Element present in the list
Enter the Option: 6
List contains some Elements
Enter the Option: 4
Now list is empty
Enter the Option: 8
List is deleted
Enter the Option: 2
List is not created
Enter the Option: 10
*****WRONG ENTRY*****
Enter the Option: 9
```

**Result:**

Thus the C program to implement List ADT by using linked list is executed successfully and the output is verified.

**Program Outcome:**

Thus the student can know to implement Linear Data Structure – List by using linked list

**Applications:**

1. It is very much useful in sorting algorithms.  
Example: Selection sort

**Viva Voce:**

1. What is list?
2. Define ADT.
3. What is linked List?
4. What are the different operations involved in List?
5. How the search operation works in list?
6. How will you find the list is full?
7. How will you find the list is empty?
8. Define pointers in c.
9. How the list used in computer memory?
10. What are the different types of linked list?

[www.FirstRanker.com](http://www.FirstRanker.com)

EX . No: 4a

**APPLICATION OF STACK****DATE****Aim:**

To write a C program to convert infix expression to postfix expression

**Algorithm:**

1. Push "(" onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
  1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
  2. Add operator to Stack.[End of If]
6. If a right parenthesis is encountered ,then:
  1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
  2. Remove the left Parenthesis.[End of If]  
[End of If]
7. END.

**Sample Output:**Enter Infix expression :  $A+(B*C-(D/E^F)*G)*H$ Postfix Expression:  $ABC*DEF^/G*-H*+F$ Enter Infix expression :  $(3^2*5)/(3*2-3)+5$ Postfix Expression:  $3^2*5*3^2*3-/5+$ **Result:**

Thus the C program to convert infix to postfix is executed successfully and the output is verified.

**Program Outcome:**

Thus the student can know to implement infix to postfix by using stack

**Ex: 4b****APPLICATION OF LIST – POLYNOMIAL ADDITION****Date:****Aim:**

To write a C program to add two polynomials

**Algorithm:**

Algorithm to add two polynomials using linked list is as follows:-

Let p and q be the two polynomials represented by linked lists

1. While p and q are not null, repeat step 2.
2. If powers of the two terms are equal  
    then if the terms do not cancel  
    then insert the sum of the terms into the sum Polynomial  
    Advance p  
    Advance q

Else if the power of the first polynomial &gt; power of second

Then insert the term from first polynomial into sum polynomial

Advance p

Else insert the term from second polynomial into sum polynomial

Advance q

3. copy the remaining terms from the non empty polynomial into the sum polynomial.

The third step of the algorithm is to be processed till the end of the polynomials has not been reached.

**Sample Output:**

Create 1st expression

Enter Coeff: 1

Enter Pow: 2

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 1st expression

The polynomial expression is:

1x<sup>2</sup>

Create 2nd expression

Enter Coeff:2

Enter Pow:3

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 2nd expression

The polynomial expression is:

$2x^3$

The polynomial expression is:

$2x^3 + 1x^2$

Add two more expressions? (Y = 1/N = 0): 0

...Program finished with exit code 0

**Result:**

Thus the C program to implement polynomial addition by using linked list is executed successfully and the output is verified.

**Program Outcome:**

Thus the student can know to implement polynomial addition by using linked list

**Viva Voce:**

1. What is Stack?
2. List the operations of stack.
3. What is linked List?
4. What is the format of infix expression?
5. What is the format of postfix expression?



**Ex: 5a**
**GRAPH REPRESENTATIONS**
**Date:**
**Aim:**

To write a C program implement adjacent matrix and adjacency list

**Algorithm:**

1. Create a graph with getting no. of vertices and no. of edges
2. Implement adjacency matrix
3. Implement adjacency list
4. Close the program

**Sample Output:**

A Program to represent a Graph by using an Adjacency Matrix method

1. Directed Graph
2. Un-Directed Graph
3. Exit

Select a proper option :

How Many Vertices ? :

Vertices 1 &amp; 2 are Adjacent ? (Y/N) : N

Vertices 1 &amp; 3 are Adjacent ? (Y/N) : Y

Vertices 1 &amp; 4 are Adjacent ? (Y/N) : Y

Vertices 2 &amp; 1 are Adjacent ? (Y/N) : Y

Vertices 2 &amp; 3 are Adjacent ? (Y/N) : Y

Vertices 2 &amp; 4 are Adjacent ? (Y/N) : N

Vertices 3 &amp; 1 are Adjacent ? (Y/N) : Y

Vertices 3 &amp; 2 are Adjacent ? (Y/N) : Y

Vertices 3 &amp; 4 are Adjacent ? (Y/N) : Y

Vertices 4 &amp; 1 are Adjacent ? (Y/N) : Y

Vertices 4 &amp; 2 are Adjacent ? (Y/N) : N

Vertices 4 &amp; 3 are Adjacent ? (Y/N) : Y

Vertex	In_Degree	Out_Degree	Total_Degree
1	2	0	2
2	1	2	3
3	0	1	1
4	1	1	2

A Program to represent a Graph by using an Adjacency List method

1. Directed Graph
2. Un-Directed Graph
3. Exit

Select a proper option :

How Many Vertices ? :

Vertices 1 & 2 are Adjacent ? (Y/N) : N  
Vertices 1 & 3 are Adjacent ? (Y/N) : Y  
Vertices 1 & 4 are Adjacent ? (Y/N) : Y  
Vertices 2 & 1 are Adjacent ? (Y/N) : Y  
Vertices 2 & 3 are Adjacent ? (Y/N) : Y  
Vertices 2 & 4 are Adjacent ? (Y/N) : N  
Vertices 3 & 1 are Adjacent ? (Y/N) : Y  
Vertices 3 & 2 are Adjacent ? (Y/N) : Y  
Vertices 3 & 4 are Adjacent ? (Y/N) : Y  
Vertices 4 & 1 are Adjacent ? (Y/N) : Y  
Vertices 4 & 2 are Adjacent ? (Y/N) : N  
Vertices 4 & 3 are Adjacent ? (Y/N) : Y

[www.FirstRanker.com](http://www.FirstRanker.com)

**Ex: 5b****GRAPH TRAVERSAL****Date:****Aim:**

To write a C program implement DFS and BFS graph traversal.

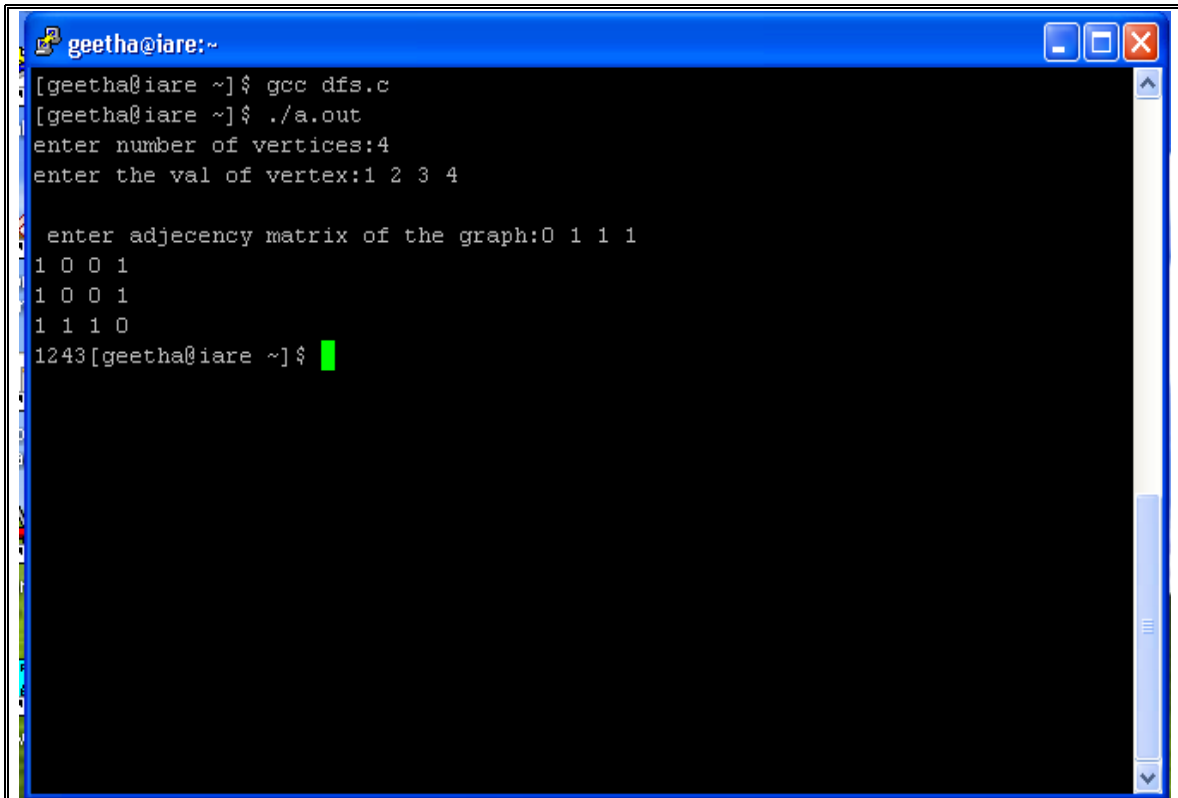
**Algorithm:****DFS**

1. Define a Stack of size total number of vertices in the graph.
2. Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the **adjacent** vertex of the vertex which is at top of the stack which is not visited and push it on to the stack.
4. Repeat step 3 until there are no new vertex to be visit from the vertex on top of the stack.
5. When there is no new vertex to be visit then use **back tracking** and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.
7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

**BFS**

1. Define a Queue of size total number of vertices in the graph.
2. Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
3. Visit all the **adjacent** vertices of the vertex which is at front of the Queue which is not visited and insert them into the Queue.
4. When there is no new vertex to be visit from the vertex at front of the Queue then delete that vertex from the Queue.
5. Repeat step 3 and 4 until queue becomes empty.
6. When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph

**Sample Output:**



```
geetha@iare:~  
[geetha@iare ~]$ gcc dfs.c  
[geetha@iare ~]$ ./a.out  
enter number of vertices:4  
enter the val of vertex:1 2 3 4  
  
enter adjacency matrix of the graph:0 1 1 1  
1 0 0 1  
1 0 0 1  
1 1 1 0  
1243[geetha@iare ~]$
```

**Result:**

Thus the C programs to implement graph representation, Adjacency matrix and Adjacency List, DFS and BFS are executed successfully and the outputs are verified.

**Program Outcome:**

Thus the student can know to implement graph representation and graph traversal.

**Viva Voce:**

1. Define Graph
2. List the operations of graph.
3. What are the different representations of graphs?
4. What are the two different graph traversals?
5. Define DFS.
6. Define BFS.
7. List the application of DFS.
8. List the application of BFS.
9. Define Indegree
10. Define Outdegree.

**Ex: 6                      IMPLEMENTATION OF SEARCHING ALGORITHMS**  
**Date                              LINEAR SEARCH AND BINARY      SEARCH****Aim:**

To write a C Program to implement different searching techniques – Linear and Binary search.

**Algorithm:****Linear Search:**

1. Read the search element from the user
2. Compare, the search element with the first element in the list.
3. If both are matching, then display "Given element found!!!" and terminate the function
4. If both are not matching, then compare search element with the next element in the list.
5. Repeat steps 3 and 4 until the search element is compared with the last element in the list.
6. If the last element in the list is also doesn't match, then display "Element not found!!!" and terminate the function.

Binary search is implemented using following steps...

1. Read the search element from the user
2. Find the middle element in the sorted list
3. Compare, the search element with the middle element in the sorted list.
4. If both are matching, then display "Given element found!!!" and terminate the function
5. If both are not matching, then check whether the search element is smaller or larger than middle element.
6. If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.
7. If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.
8. Repeat the same process until we find the search element in the list or until sublist contains only one element.
9. If that element also doesn't match with the search element, then display "Element not found in the list!!!" and terminate the function.

**Sample Output:****Linear Search**

Enter size of the list: 4

Enter any

4 integer values: 4 8 1 9

Enter the element to be Search: 9

Element is found at 3 index

...Program finished with exit code 10

### Binary Search

Enter the size of the list: 4

Enter integer values in Ascending order

Enter 4 integers

1 6 9 12

Enter value to be search: 9

Element found at index 2.

### Result:

Thus the C programs to implement Linear search and binary search are executed successfully and output are verified.

### Program Outcome:

Thus the student can know to implement graph representation and graph traversal.

### Viva Voce:

1. Define Search
2. Define Sequential Search.
3. What is Linear Search?
4. What is the time complexity of linear search?
5. What is Binary Search?
6. What is the time complexity of linear search?
7. List the application of Linear Search.
8. List the application of Binary Search.

**Ex: 7 IMPLEMENTATION OF SORTING ALGORITHMS – BUBBLE SORT, RADIX SORT, SHELL SORT**

**Date:**

**Aim:**

To write a C program to implement different sorting algorithms - Bubble Sort, Radix Sort, Shell Sort

**Algorithm:**

**Bubble Sort:**

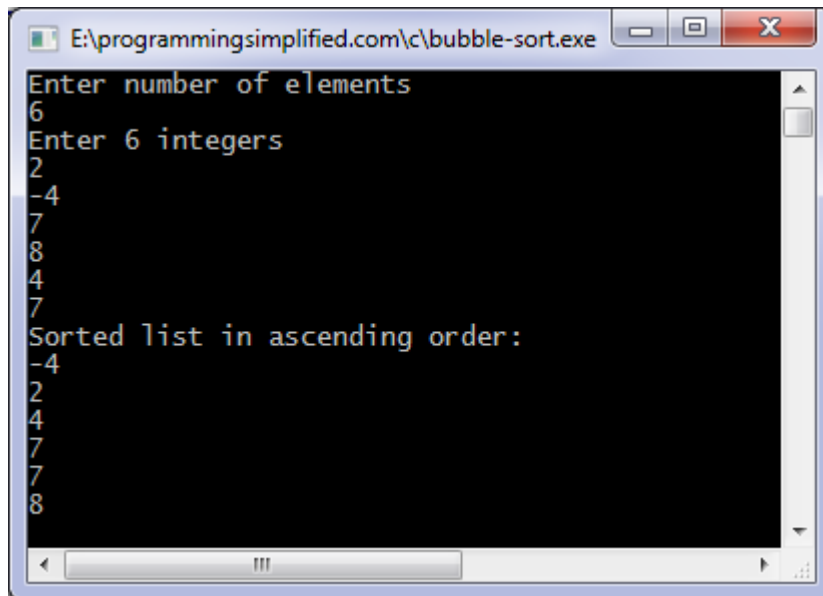
1. Read the value of n
2. Get the inputs for array[n].
3. Implement Bubble sort
  - Step 1: Repeat Steps 2 and 3 for i=1 to 10
  - Step 2: Set j=1
  - Step 3: Repeat while j<=n
    - (A) if a[i] < a[j]  
Then interchange a[i] and a[j]  
[End of if]
    - (B) Set j = j+1
  - [End of Inner Loop]
  - [End of Step 1 Outer Loop]
  - Step 4: Exit
4. Stop the program

**Shell Sort**

1. Read the value of n
2. Get the inputs for array[n].
3. Implement shell sort
4. Stop the program

**Radix Sort**

1. Read the value of n
2. Get the inputs for array[n].
3. Implement radix sort
4. Stop the program

**Sample Output****Bubble sort**

```
E:\programmingsimplified.com\c\bubble-sort.exe
Enter number of elements
6
Enter 6 integers
2
-4
7
8
4
7
Sorted list in ascending order:
-4
2
4
7
7
8
```

**Shell sort**

```
Enter total no. of elements : 10
Enter numbers : 36 432 43 44 57 63 94 3 5 6
Sorted array is : 3 5 6 36 43 44 57 63 94 432
```

**Radix sort**

```
$ gcc RadixSort.c
$ ./a.out
Enter total no. of elements : 8
Enter numbers : 802 2 24 45 66 75 90 170
Sorted array is : 2 24 45 66 75 90 170 802
```

**Result:**

Thus the C programs to implement different sorting algorithms are executed successfully and outputs are verified.

**Program Outcome:**

Thus the student can know to implement different sorting algorithms.

**Viva Voce:**

1. What is sorting?
2. What are the different types of sorting?
3. What is Bubble sort?
4. What is Radix sort?
5. What is Shell sort?
6. What is insertion sort?



7. Mention about Selection sort.
8. Compare Bubble and Radix sort.
9. Compare Shell sort and Bubble sort
10. Compare Shell and Radix sort

[www.FirstRanker.com](http://www.FirstRanker.com)

**Ex. No: 8****IMPLEMENTATION OF HEAP****DATE:****Aim:**

To write a Cprogram to implement heap data structure

**Algorithm:**

1. Start the program
2. Initially, declare n=0 for number of nodes in the heap'
3. By using while, create a menu for all operations of heaps
4. Perform all the operation and display the result
5. Stop the program

**Output:**

```
$ cc pgm66.c
$ a.out
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 1
Enter the element to be inserted to the list : 30
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 1
Enter the element to be inserted to the list : 50
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 1
Enter the element to be inserted to the list : 70
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 2
Enter the elements to be deleted from the list: 10
10 not found in heap list
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 2
Enter the elements to be deleted from the list: 50
1.Insert the element
2.Delete the element
```

```
3.Display all elements
4.Quit
Enter your choice : 1
Enter the element to be inserted to the list : 100
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 3
100 30 70
1.Insert the element
2.Delete the element
3.Display all elements
4.Quit
Enter your choice : 4
```

**Result:**

Thus the C programs to implement heap is executed successfully and output is verified.

**Program Outcome:**

Thus the student can know to implement heap data structure.

**Viva Voce:**

1. What is heap?
2. What are called heap property?
3. List the operations of heap.
4. What is order property?
5. What is Shape property?
6. What is Priority queue?
7. List the applications of heap.
8. List the types of heap.

**Ex. No: 9****IMPLEMENTATION OF BINARY TREE & ITS OPERATIONS****DATE:****Aim:**

To write a Cprogram to implement binary tree and its operation

**Algorithm:**

1. Start the program
2. Create a tree
3. Perform the Insertion, Deletion and Search operation
4. Display Preorder, Postorder, Inorder traversal data
5. Stop the program

**Sample Output**

&gt; IN ORDER -

```
> 0 2 2 8 9 9 16 17 18 19 19 20 20 21 21 22 23 26 28 29 35 37 40 41 45 50 51 54 62 63 6
3 64 66 66 68 69 7
0 71 73 75 76 76 78 79 79 80 83 83 85 86 89 91 91 95 96 96 101 101 101 102 103 104 10
4 104 106 107 108 108 109 111 112
113 114 115 115 115 116 117 117 119 120 123 124 125 127 127 129 129 129 130 130 1
34 134 137 139 139 140 142 143 143
```

&gt; PRE ORDER -

```
> 127 101 21 21 2 2 0 8 20 18 16 9 9 17 19 19 20 96 51 22 35 26 23 28 29 37 45 40 41 5
0 79 76 66 66 63 5
4 63 62 64 73 68 69 71 70 76 75 79 78 89 86 80 83 83 85 91 91 95 96 101 101 116 108 1
02 107 106 104 104 103 104 108 11
3 112 109 111 115 115 115 114 120 117 117 119 127 125 124 123 129 129 129 140 137
134 134 130 130 139 139 143 142 143
```

&gt; POST ORDER -

```
> 0 2 9 9 17 16 19 20 19 18 20 8 2 21 23 29 28 26 41 40 50 45 37 35 22 62 63 54 64 63 6
6 70 71 69 68 75
76 73 66 78 79 76 83 85 83 80 86 91 96 95 91 89 79 51 101 101 96 21 104 103 104 104
106 108 107 102 111 109 112 114 1
15 115 115 113 108 117 119 117 123 124 125 127 120 116 101 129 129 130 130 134 134
139 139 137 143 142 143 140 129 127
```

&gt; TEST SEARCH:

```
- 2
- 8
```

- 9  
- 16  
- 17  
- 18  
- 19  
- 20  
- 21  
- 22  
- 23  
- 26  
- 28  
- 29  
- 35  
- 37  
- 40  
- 41  
- 45  
- 50  
- 51  
- 54  
- 62  
- 63  
- 64  
- 66  
- 68  
- 69  
- 70  
- 71  
- 73  
- 75  
- 76  
- 78  
- 79  
- 80  
- 83  
- 85  
- 86  
- 89  
- 91  
- 95  
- 96

<SUCCESS> = 43 <FAILED> = 57

...Program finished with exit code 0

**Result:**

Thus the C programs to implement Binary tree is executed successfully and output is verified.

**Program Outcome:**

Thus the student can know to implement binary tree.

[www.FirstRanker.com](http://www.FirstRanker.com)

**Ex. No: 10****IMPLEMENTATION OF BINARY SEARCH TREE****DATE****Aim:**

To write a Cprogram to implement binary search tree

**Algorithm:**

1. Start the program
2. Create a tree
3. Perform the Insertion, Deletion and Search operation
4. Display Preorder, Postorder, Inorder traversal data
5. Stop the program

**Sample Output:**

```
cc tree43.c
$ a.out
OPERATIONS ---
1 - Insert an element into tree
2 - Delete an element from the tree
3 - Inorder Traversal
4 - Preorder Traversal
5 - Postorder Traversal
6 - Exit

Enter your choice : 1
Enter data of node to be inserted : 40

Enter your choice : 1
Enter data of node to be inserted : 20

Enter your choice : 1
Enter data of node to be inserted : 10

Enter your choice : 1
Enter data of node to be inserted : 30

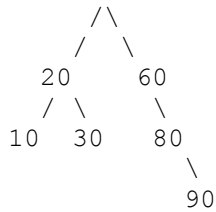
Enter your choice : 1
Enter data of node to be inserted : 60

Enter your choice : 1
Enter data of node to be inserted : 80

Enter your choice : 1
Enter data of node to be inserted : 90

Enter your choice : 3
10 -> 20 -> 30 -> 40 -> 60 -> 80 -> 90 ->
```

40

**Result:**

Thus the C programs to implement Binary search tree is executed successfully and output is verified.

**Program Outcome:**

Thus the student can know to implement binary search tree.

**Viva Voce:**

1. What is tree?
2. What are called binary tree?
3. List the operations of binary tree.
4. What are the condition for left sub tree and right sub tree?
5. What are called tree traversal?
6. What is In-order?
7. What is preorder?
8. What is postorder?
9. What are called binary search tree?
10. Compare Binary tree and Binary Search tree.



**Ex. No: 11****HASHING TECHNIQUES****DATE****Aim:**

To write a C program to implement hash table

**Algorithm:**

1. Create a structure, data (hash table item) with key and value as data.
2. Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).
3. A menu is displayed on the screen.
4. User must choose one option from four choices given in the menu
5. Perform all the operations
6. Stop the program

**Sample Output**

Implementation of Hash Table in C

MENU:-

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice:- 3

Size of Hash Table is:- 0

Do you want to continue:- (press 1 for yes) 1

Implementation of Hash Table in C

MENU:-

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice:- 1

Inserting element in Hash Table

Enter key :- 1

key (1) and value (1) has been inserted

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C chaining with singly linked list

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 4

array[0] has no elements

array[1] has no elements      key = 1 and value = 1

array[2] has no elements

array[3] has no elements

array[4] has no elements

array[5] has no elements

array[6] has no elements

array[7] has no elements

array[8] has no elements

array[9] has no elements

\*\*\*\*\*

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table
4. Display Hash Table

Please enter your choice-: 2

Deleting key from Hash Table

Enter the key to delete-: 3

This key does not exist.

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C

MENU-:

1. Inserting item in the Hash Table
2. Removing item from the Hash Table
3. Check the size of Hash Table

#### 4. Display Hash Table

Please enter your choice-: 2

Deleting key from Hash Table

Enter the key to delete-: 1

Key 1 has been removed.

Do you want to continue-:(press 1 for yes) 2

#### **Result:**

Thus the C programs to implement hash table is executed successfully and output is verified.

#### **Program Outcome:**

Thus the student can know to implement hash table.

#### **Viva Voce:**

1. What is hash?
2. What are called bucket?
3. What is key?
4. What are called hash function.
5. List the different hashing techniques?
6. What is Open Addressing?
7. What is Linear Probing?