Laboratory 8

1. Questions

a) Implement a linked list and perform following operations.
I. Insert a node before and after a given node
II. Delete a node before and after a given node
b) Implement a linked list to create and print a binary tree.

2. Algorithm

1.Algorithm to to implement Insert a node before and after a given node
Delete a node before and after a given node

Inserting At Beginning of the list
Step 1: Create a newNode with given value.
Step 2: Check whether list is Empty (head == NULL)
Step 3: If it is Empty then, set newNode→next = NULL and head = newNode.
Step 4: If it is Not Empty then, set newNode→next = head and head = newNode.

Inserting At End of the list
Step 1: Create a newNode with given value and newNode → next as NULL.
Step 2: Check whether list is Empty (head == NULL).
Step 3: If it is Empty then, set head = newNode.
Step 4: If it is Not Empty then, define a node pointer temp and initialize with head.
Step 5: Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
Step 6: Set temp → next = newNode.

Inserting At Specific location in the list (After a Node)
Step 1: Create a newNode with given value.
Step 2: Check whether list is Empty (head == NULL)
Step 3: If it is Empty then, set newNode → next = NULL and head = newNode.
Step 4: If it is Not Empty then, define a node pointer temp and initialize with head.
Step 5: Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
Step 6: Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
Step 7: Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

Deleting a Specific Node from the list
Step 1: Check whether list is Empty (head == NULL)
Step 2: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
Step 3: If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
Step 4: Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.
Step 5: If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.
Step 6: If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
Step 7: If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).
Step 8: If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head).
Step 9: If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.

Step 10: If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL).
Step 11: If temp1 is last node then set temp2 → next = NULL and delete temp1 (free(temp1)).
Step 12: If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...
Step 1: Check whether list is Empty (head == NULL)
Step 2: If it is Empty then, display 'List is Empty!!!' and terminate the function.
Step 3: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
Step 4: Keep displaying temp → data with an arrow (--->) until temp reaches to the last node
Step 5: Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

2.**Algorithm to program to Implement a linked list to create and print a binary tree.**

Step 1=Define Node class which has three attributes namely: **data left** and **right**. Here, left represents the left child of the node and right represents the right child of the node.

Step 2=When a node is created, data will pass to data attribute of the node and both left and right will be set to **null**.

Step 3=Define another class which has an attribute root.

a. **Root** represents the root node of the tree and initialize it to null.

step 4=insert() will add a new node to the tree:

It checks whether the root is null, which means the tree is empty. It will add the new node as root.

b. Else, it will add root to the queue.
c. The variable node represents the current node.
d. First, it checks whether a node has a left and right child. If yes, it will add both nodes to queue.
e. If the left child is not present, it will add the new node as the left child.
f. If the left is present, then it will add the new node as the right child.

Step 5=Inorder() will display nodes of the tree in inorder fashion.

a.It traverses the entire tree then prints out left child followed by root then followed by the right child.

3. Program

main.c saved

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();


struct node
{
    int info;
    struct node *next;
};
struct node *start=NULL;
int main()
{
    int choice;
    while(1){

        printf("                MENU
        printf("\n 1.Create     ");
        printf("\n 2.Display    ");
        printf("\n 3.Insert at the beginning    ");
        printf("\n 4.Insert at the end  ");
        printf("\n 5.Insert at specified position       ");
        printf("\n 6.Delete from beginning      ");
        printf("\n 7.Delete from the end        ");
        printf("\n 8.Delete from specified position     ");
        printf("\n 9.Exit       ");
        printf("\n-------------------------------------\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                create();
                break;
            case 2:
                display();
                break;
            case 3:
                insert_begin();
                break;
            case 4:
                insert_end();
                break;
            case 5:
                insert_pos();
                break;
            case 6:
                delete_begin();
                break;
            case 7:
                delete_end();
                break;
            case 8:
                delete_pos();
                break;

            case 9:
                exit(0);
                break;

            default:
                printf("\n Wrong Choice:\n");
```

```
                                                    break;
                                                }
                                            }
                                            return 0;
                                        }
                                        void create()
                                        {
                                            struct node *temp,*ptr;
                                            temp=(struct node *)malloc(sizeof(struct node));
                                            if(temp==NULL)
                                            {
                                                printf("Out of Memory Space:\n");
                                                exit(0);
                                            }
                                            printf("Enter the data value for the node:");
                                            scanf("%d",&temp->info);
                                            temp->next=NULL;
                                            if(start==NULL)
                                            {
                                                start=temp;
                                            }
                                            else
                                            {
                                                ptr=start;
                                                while(ptr->next!=NULL)
                                                {
                                                    ptr=ptr->next;
                                                }
                                                ptr->next=temp;
                                            }
                                        }
                                        void display()
                                        {
                                            struct node *ptr;
                                            if(start==NULL)
```

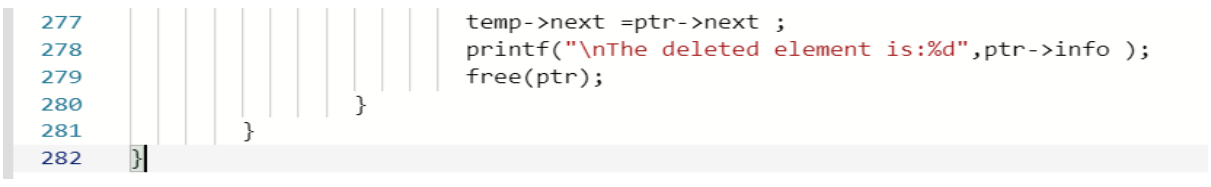
```
                                                printf("\nList is empty:");
                                                return;
                                            }
                                            else
                                            {
                                                ptr=start;
                                                printf("\nThe List elements are \n:");
                                                while(ptr!=NULL)
                                                {
                                                    printf("%d",ptr->info );
                                                    ptr=ptr->next ;
                                                }
                                            }
                                        }
                                        void insert_begin()
                                        {
                                            struct node *temp;
                                            temp=(struct node *)malloc(sizeof(struct node));
                                            if(temp==NULL)
                                            {
                                                printf("\nOut of Memory Space:");
                                                return;
                                            }
                                            printf("\nEnter the data value for the node:" );
                                            scanf("%d",&temp->info);
                                            temp->next =NULL;
                                            if(start==NULL)
                                            {
                                                start=temp;
                                            }
                                            else
                                            {
                                                temp->next=start;
                                                start=temp;
                                            }
```

```
142  }
143  void insert_end()
144  {
145      struct node *temp,*ptr;
146      temp=(struct node *)malloc(sizeof(struct node));
147      if(temp==NULL)
148      {
149          printf("\nOut of Memory Space:");
150          return;
151      }
152      printf("\nEnter the data value for the node:" );
153      scanf("%d",&temp->info );
154      temp->next =NULL;
155      if(start==NULL)
156      {
157          start=temp;
158      }
159      else
160      {
161          ptr=start;
162          while(ptr->next !=NULL)
163          {
164              ptr=ptr->next ;
165          }
166          ptr->next =temp;
167      }
168  }
169  void insert_pos()
170  {
171      struct node *ptr,*temp;
172      int i,pos;
173      temp=(struct node *)malloc(sizeof(struct node));
174      if(temp==NULL)
175      {
```

```
175      {
176          printf("\nOut of Memory Space:");
177          return;
178      }
179      printf("\nEnter the position for the new node to be inserted:");
180      scanf("%d",&pos);
181      printf("nEnter the data value of the node:t");
182      scanf("%d",&temp->info) ;
183
184      temp->next=NULL;
185      if(pos==0)
186      {
187          temp->next=start;
188          start=temp;
189      }
190      else
191      {
192          for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;
193              if(ptr==NULL)
194              {
195                  printf("\nPosition not found:[Handle with care]");
196                  return;
197              }
198          }
199          temp->next =ptr->next ;
200          ptr->next=temp;
201      }
202  }
203  void delete_begin()
204  {
205      struct node *ptr;
206      if(ptr==NULL)
207      {
```

```
208                     printf("\n List is Empty:");
209                     return;
210                 }
211                 else
212                 {
213                     ptr=start;
214                     start=start->next ;
215                     printf("\nThe deleted element is :%d",ptr->info);
216                     free(ptr);
217                 }
218     }
219     void delete_end()
220     {
221                 struct node *temp,*ptr;
222                 if(start==NULL)
223                 {
224                     printf("\nList is Empty:");
225                     exit(0);
226                 }
227                 else if(start->next ==NULL)
228                 {
229                     ptr=start;
230                     start=NULL;
231                     printf("\nThe deleted element is:%d",ptr->info);
232                     free(ptr);
233                 }
234                 else
235                 {
236                     ptr=start;
237                     while(ptr->next!=NULL)
238                     {
239                         temp=ptr;
240                         ptr=ptr->next;
241                     }
242                     temp->next=NULL;
```

```
243                     printf("\nThe deleted element is:%d",ptr->info);
244                     free(ptr);
245             }
246     }
247     void delete_pos()
248     {
249         int i,pos;
250         struct node *temp,*ptr;
251         if(start==NULL)
252         {
253             printf("\nThe List is Empty:");
254             exit(0);
255         }
256         else
257         {
258             printf("\nEnter the position of the node to be deleted:");
259             scanf("%d",&pos);
260             if(pos==0)
261             {
262                 ptr=start;
263                 start=start->next ;
264                 printf("\nThe deleted element is:%d",ptr->info  );
265                 free(ptr);
266             }
267             else
268             {
269                 ptr=start;
270                 for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;
271                     if(ptr==NULL)
272                     {
273                         printf("nPosition not Found:\n");
274                         return;
275                     }
276                 }
```



```
277                     temp->next =ptr->next ;
278                     printf("\nThe deleted element is:%d",ptr->info );
279                     free(ptr);
280                 }
281             }
282     }
```

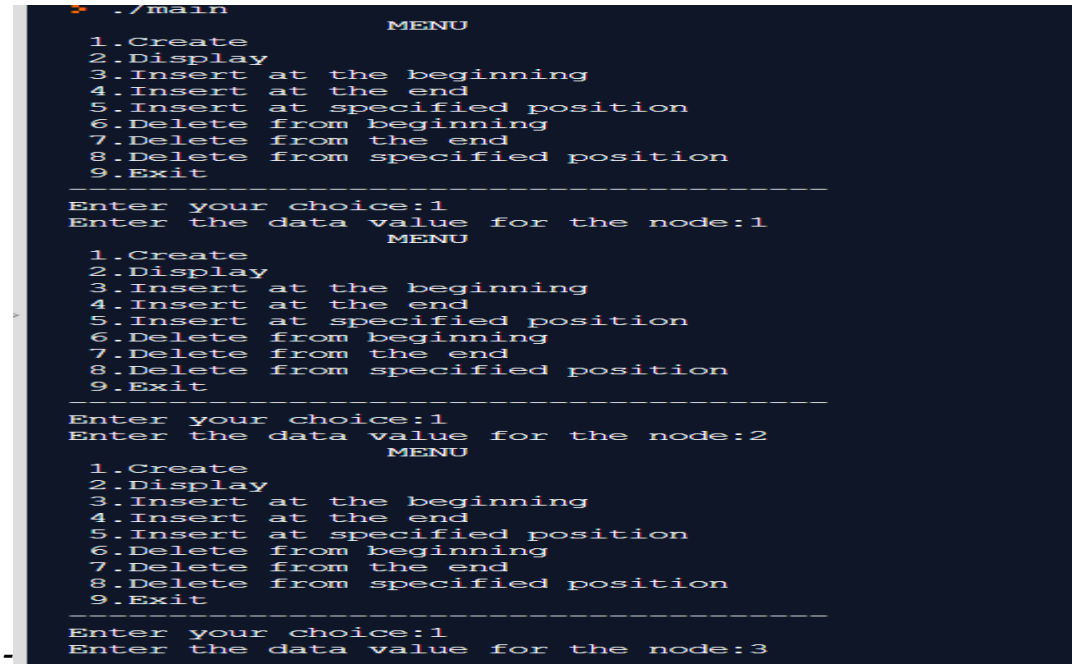
Fig 1 program to implement Insert a node before and after a given node
Delete a node before and after a given node

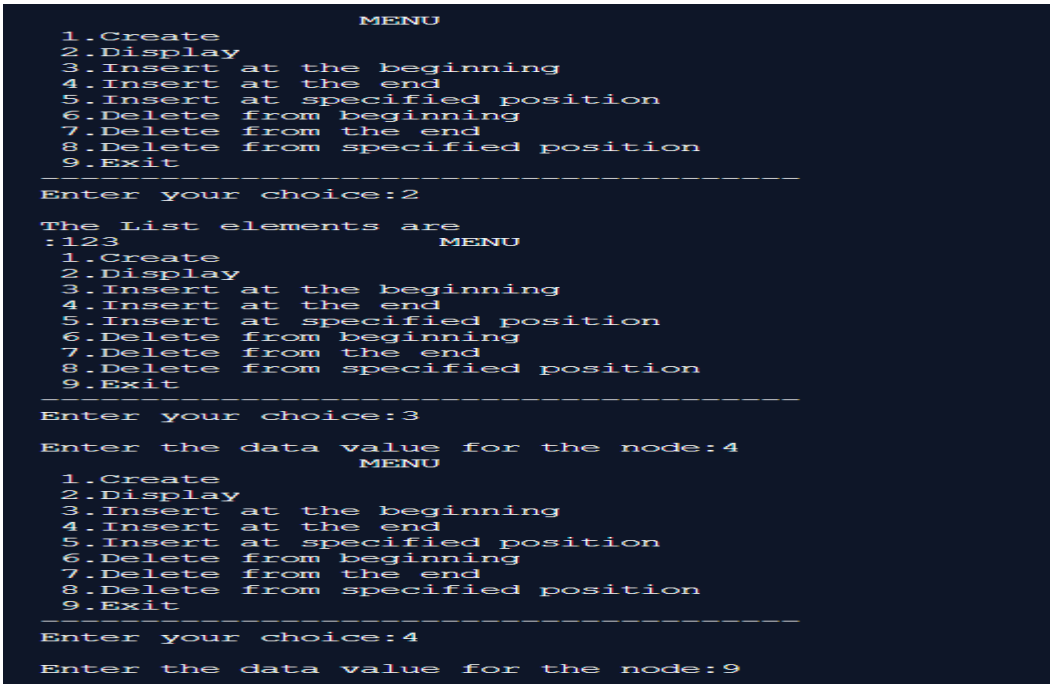
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

//Represent a node of binary tree
struct node{
    int data;
    struct node *left;
    struct node *right;
};

//Represent the root of binary tree
struct node *root = NULL;

//createNode() will create a new node
struct node* createNode(int data){
    //Create a new node
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    //Assign data to newNode, set left and right child to NULL
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

//Represent a queue
struct queue
{
{
    int front, rear, size;
    struct node* *arr;
};

//createQueue() will create a queue
struct queue* createQueue()
{
    struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));

    newQueue->front = -1;
    newQueue->rear = 0;
    newQueue->size = 0;

    newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ));

    return newQueue;
}

//Adds a node to queue
void enqueue(struct queue* queue, struct node *temp){
    queue->arr[queue->rear++] = temp;
    queue->size++;
}

//Deletes a node from queue
struct node *dequeue(struct queue* queue){
    queue->size--;
    return queue->arr[++queue->front];
}
```

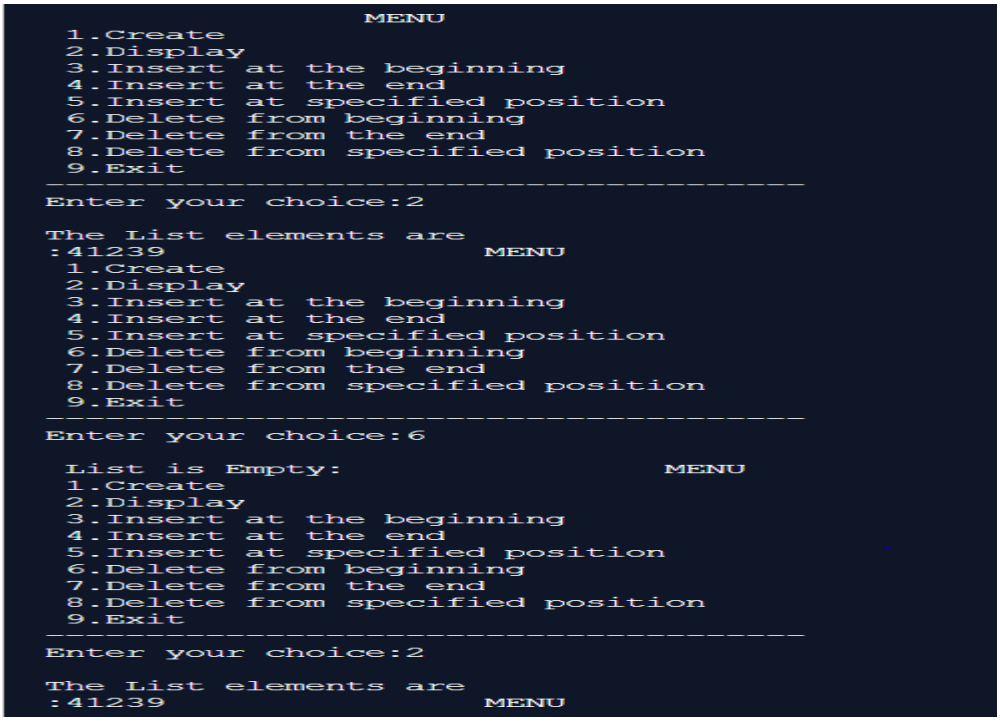
```
//insertNode() will add new node to the binary tree
void insertNode(int data) {
   //Create a new node
   struct node *newNode = createNode(data);
   //Check whether tree is empty
   if(root == NULL){
      root = newNode;
      return;
   }
   else {
      //Queue will be used to keep track of nodes of tree level-wise
      struct queue* queue = createQueue();
      //Add root to the queue
      enqueue(queue, root);

      while(true) {
         struct node *node = dequeue(queue);
         //If node has both left and right child, add both the child to queue
         if(node->left != NULL && node->right != NULL) {
            enqueue(queue, node->left);
            enqueue(queue, node->right);
         }
         else {
            //If node has no left child, make newNode as left child
            if(node->left == NULL) {
               node->left = newNode;
               enqueue(queue, node->left);
            }
            //If node has left child but no right child, make newNode as right child
            else {

               node->right = newNode;
               enqueue(queue, node->right);
            }
            break;
         }
      }
   }
}

//inorder() will perform inorder traversal on binary search tree
void inorderTraversal(struct node *node) {
   //Check whether tree is empty
   if(root == NULL){
      printf("Tree is empty\n");
      return;
   }
   else {

      if(node->left != NULL)
         inorderTraversal(node->left);
      printf("%d ", node->data);
      if(node->right != NULL)
         inorderTraversal(node->right);

      }
   }

int main(){

   //Add nodes to the binary tree

   //Add nodes to the binary tree
   insertNode(1);
   //1 will become root node of the tree
   printf("Binary tree after insertion: \n");
   //Binary after inserting nodes
   inorderTraversal(root);

   insertNode(2);
   insertNode(3);
   //2 will become left child and 3 will become right child of root node 1
   printf("\nBinary tree after insertion: \n");
   //Binary after inserting nodes
   inorderTraversal(root);

   insertNode(4);
   insertNode(5);
   //4 will become left child and 5 will become right child of node 2
   printf("\nBinary tree after insertion: \n");
   //Binary after inserting nodes
   inorderTraversal(root);

   insertNode(6);
   insertNode(7);
   //6 will become left child and 7 will become right child of node 3
   printf("\nBinary tree after insertion: \n");
   //Binary after inserting nodes
   inorderTraversal(root);

   return 0;
```

Fig 2 program to Implement a linked list to create and print a binary tree.

4. Presentation of Results







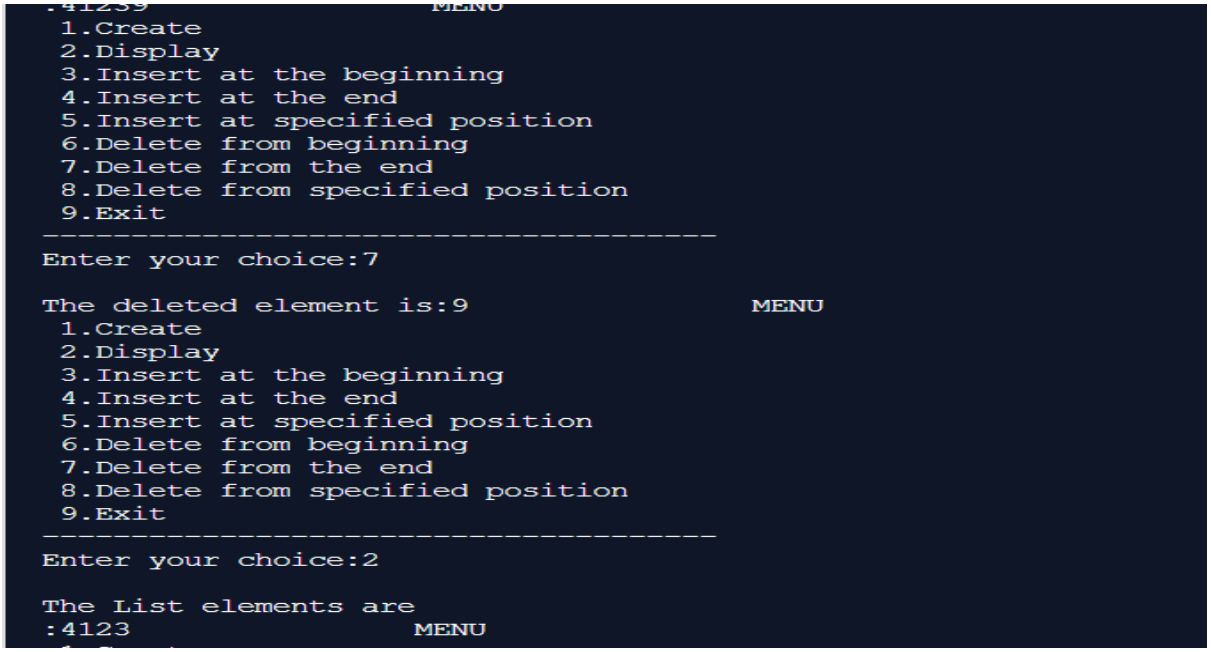


Fig 3 Result of 1 program to implement Insert a node before and after a given node
Delete a node before and after a given node

```
Binary tree after insertion
1
Binary tree after insertion
2 1 3
Binary tree after insertion
4 2 5 1 3
Binary tree after insertion
4 2 5 1 6 3 7
```

Fig 4 Result of program to Implement a linked list to create and print a binary tree.

5. Conclusions

In this lab we learnt to Implement a linked list and perform Insertion of node before and after a given node and Deletetion of a node before and after a given node and to Implement a linked list to create and print a binary tree.