RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

25 years OF ACADEMIC EXCELLENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Laboratory Manual

REGULATION 2023

CS23231 – DATA STRUCTURES

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution, Affiliated to Anna University Rajalakshmi Nagar, Thandalam – 602 105

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS23231 – DATA STRUCTURES

(*Regulation 2023*)

LAB MANUAL

Name	: . . . . . Monisha KR . . . . .
Register No.	: . 2116231801111. . . . .
Year / Branch / 1Section	: . . . .I-YEAR ARTIFICIALINTELLIGENCEANDDATASCIENCE.
Semester	: . . . . . II semester. . . . .
Academic Year	: 2023-2024. . . . .

LESSON PLAN

Course Code	Course Title (Laboratory Integrated Theory Course)	L	T	P	C
CS23231	Data Structures	3	0	4	5

LIST OF EXPERIMENTS	
Sl. No	**Name of the experiment**
Week 1	Implementation of Single Linked List (Insertion, Deletion and Display)
Week 2	Implementation of Doubly Linked List (Insertion, Deletion and Display)
Week 3	Applications of Singly Linked List (Polynomial Manipulation)
Week 4	Implementation of Stack using Array and Linked List implementation
Week 5	Applications of Stack (Infix to Postfix)
Week 6	Applications of Stack (Evaluating Arithmetic Expression)
Week 7	Implementation of Queue using Array and Linked List implementation
Week 8	Implementation of Binary Search Tree
Week 9	Performing Tree Traversal Techniques
Week 10	Implementation of AVL Tree
Week 11	Performing Topological Sorting
Week 12	Implementation of BFS, DFS
Week 13	Implementation of Prim's Algorithm
Week 14	Implementation of Dijkstra's Algorithm
Week 15	Program to perform Sorting
Week 16	Implementation of Open Addressing (Linear Probing and Quadratic Probing)
Week 17	Implementation of Rehashing

INDEX

S. No.	Name of the Experiment	Expt. Date	Page No	Faculty Sign
1	Implementation of Single Linked List (Insertion, Deletion and Display)			
2	Implementation of Doubly Linked List (Insertion, Deletion and Display)			
3	Applications of Singly Linked List (Polynomial Manipulation)			
4	Implementation of Stack using Array and Linked List implementation			
5	Applications of Stack (Infix to Postfix)			
6	Applications of Stack (Evaluating Arithmetic Expression)			
7	Implementation of Queue using Array and Linked List implementation			
8	Performing Tree Traversal Techniques			
9	Implementation of Binary Search Tree			
10	Implementation of AVL Tree			
11	Performing Topological Sorting			
12	Implementation of BFS, DFS			
13	Implementation of Prim's Algorithm			
14	Implementation of Dijkstra's Algorithm			
15	Program to perform Sorting			
16	Implementation of Collision Resolution Techniques			

Note: Students have to write the Algorithms at left side of each problem statements.

Ex. No.:01	Implementation of Single Linked List	Date:29/02/2024

Write a C program to implement the following operations on Singly Linked List.

(i) Insert a node in the beginning of a list.

(ii) Insert a node after P

(iii) Insert a node at the end of a list

(iv) Find an element in a list

(v) FindNext

(vi) FindPrevious

(vii) isLast

(viii) isEmpty

(ix) Delete a node in the beginning of a list.

(x) Delete a node after P

(xi) Delete a node at the end of a list

(xii) Delete the List

Algorithm:

PROGRAM:

```
#include <stdio.h>
#include<stdlib.h>
void createfnode(int ele);
void insertfront(int ele);
void insertend(int ele);
void display();
//type declaration of a node
struct node
{
int data;
struct node* next;
};
struct node* head = NULL;
struct node *newnode;
void insertfront(int ele)
{
newnode=(struct node*)malloc(sizeof(struct node));
if(newnode!=NULL)
{ newnode->data=ele;
if(head!=NULL)
{
newnode->next=head;
head=newnode;
}
else
{

newnode->next=NULL;
head=newnode;

}
}
}

void insertend(int ele)
{
newnode=(struct node*)malloc(sizeof(struct node));
if(newnode!=NULL)
{
newnode->data=ele;
newnode->next=NULL;
if(head!=NULL)
{
struct node *t;
t=head;
while(t->next!=NULL)
{
t=t->next;
}
newnode->next=NULL;
t->next=newnode;
}
```

```
else
{
head=newnode;
}
}
}
int listsize()
{
int c=0;
struct node *t;
t=head;
while(t!=NULL)
{
c=c+1;
t=t->next;
}
printf("\n The size of the list is %d:\n",c);
return c;
}
void insertpos(int ele,int pos)
{
int ls=0;
ls=listsize();
if(head == NULL && (pos <= 0 || pos > 1))
{
printf("\nInvalid position to insert a node\n");
return;
}
// if the list is not empty and the position is out of range
if(head != NULL && (pos <= 0 || pos > ls))
{
printf("\nInvalid position to insert a node\n");
return;
}
struct node* newnode = NULL;
newnode=(struct node*)malloc(sizeof(struct node));
if(newnode != NULL)
{
newnode->data=ele;
struct node* temp = head;
//getting the position-1 node
int count = 1;
while(count < pos-1)
{
temp = temp -> next;
count += 1;
}
//if the position is 1 then insertion at the beginning
if(pos == 1)
{
newnode->next = head;
head = newnode;
}
else
{
newnode->next = temp->next;
temp->next = newnode;
```

```
}
}
}
void findnext(int s)
{
struct node *temp;
temp=head;
if(temp==NULL&&temp->next==NULL)
{
printf("No next element ");
}
else
{
while(temp->data!=s)
{
temp=temp->next;
}
printf("\nNext Element of %d is %d\n",s,temp->next->data);
}
}
void findprev(int s)
{
struct node *temp;
temp=head;
if(temp==NULL)
{
printf("List is empty ");
}
else
{
while(temp->next->data!=s)
{
temp=temp->next;
}
printf("\n The previous ele of %d is %d\n",s,temp->data);
}
}
void find(int s)
{
struct node *temp;
temp=head;
if(head==NULL)
{
printf("\n List is empty");
}
else
{
while(temp->data!=s && temp->next!=NULL)
{
temp=temp->next;
}
if(temp!=NULL && temp->data==s)
{
printf("\n Searching ele %d is present in the addr of %p",temp-
>data,temp);
}
else
{
```

```
printf("\n Searching elem %d is not present",s);
}
}
}
void isempty()
{
if(head==NULL)
{
printf("\nList is empty\n");
}
else
{
printf("\nList is not empty\n");
}
}

void deleteAtBeginning()
{
struct node *t;
t=head;
head=t->next;
}
void deleteAtEnd()
{
struct node *temp;
temp=head;
if(head==NULL)
{
printf("\n List is empty");
}
else
{
while(temp->next->next!=NULL)
{
temp=temp->next;
}
temp->next=NULL;
}

}
void display()
{
struct node *t;
t=head;
while(t!=NULL)
{
printf("%d\t",t->data);
t=t->next;
}
}
void delete(int ele)
{
struct node *t;
t=head;
if(t->data==ele)
{
head=t->next;
}
```

```
else
{
while(t->next->data!=ele)
{
t=t->next;
}
t->next=t->next->next;
}
}
int main()
{
do
{
int ch,a,pos;
printf("\n Choose any one operation that you would like to perform\n");
printf("\n 1.Insert the element at the beginning");
printf("\n 2.Insert the element at the end");
printf("\n 3. To insert at the specified position");
printf("\n 4. To view list");
printf("\n 5.To view list size");
printf("\n 6.To delete first element");
printf("\n 7.To delete last element");
printf("\n 8.To find next element");
printf("\n 9. To find previous element");
printf("\n 10. To find search for an element");
printf("\n 11. To quit");
printf("\n Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n Insert an element to be inserted at the beginning\n");
scanf("%d",&a);
insertfront(a);
break;
case 2:
printf("\n Insert an element to be inserted at the End\n");
scanf("%d",&a);
insertend(a);
break;
case 3:
printf("\n Insert an element and the position to insert in the list\n");
scanf("%d%d",&a,&pos);
insertpos(a,pos);
break;
case 4:
display();
break;
case 5:
listsize();
break;
case 6:
printf("\n Delete an element to be in the beginning\n");
deleteAtBeginning();
break;
case 7:
printf("\n Delete an element to be at the end\n");
deleteAtEnd();
break;
```

```
case 8:
printf("\n enter the element to which you need to find next ele in the
list\n");;
scanf("%d",&a);
findnext(a);
break;
case 9:
printf("\n enter the element to which you need to find prev ele in the
list\n");
scanf("%d",&a);
findprev(a);
break;
case 10:
printf("\n enter the element to find the address of it\n");
scanf("%d",&a);
find(a);
break;
case 11:
printf("Ended");
exit(0);
default:
printf("Invalid option is chosen so the process is quit");
}
}while(1);
return 0;
}
```

OUTPUT:

```
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
4
2       5       3
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
5

 The size of the list is 3:

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
6

 Delete an element to be in the beginning
```

```
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
6

 Delete an element to be in the beginning

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
7

 Delete an element to be at the end

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
8

 enter the element to which you need to find next ele in the list
2
```

```
Choose any one operation that you would like to perform

1.Insert the element at the beginning
2.Insert the element at the end
3. To insert at the specified position
4. To view list
5.To view list size
6.To delete first element
7.To delete last element
8.To find next element
9. To find previous element
10. To find search for an element
11. To quit
Enter your choice
1

Insert an element to be inserted at the beginning
2

Choose any one operation that you would like to perform

1.Insert the element at the beginning
2.Insert the element at the end
3. To insert at the specified position
4. To view list
5.To view list size
6.To delete first element
7.To delete last element
8.To find next element
9. To find previous element
10. To find search for an element
11. To quit
Enter your choice
2

Insert an element to be inserted at the End
3

Choose any one operation that you would like to perform

1.Insert the element at the beginning
2.Insert the element at the end
3. To insert at the specified position
4. To view list
5.To view list size
6.To delete first element
7.To delete last element
8.To find next element
9. To find previous element
10. To find search for an element
11. To quit
Enter your choice
3

Insert an element and the position to insert in the list
5
2

The size of the list is 2:
```

RESULT: Thus the program was successfully executed.

Ex. No.: 02	Implementation of Doubly Linked List	Date:07/03/2024

Write a C program to implement the following operations on Doubly Linked List.

(i) Insertion

(ii) Deletion

(iii) Search

(iv) Display

Algorithm:

PROGRAM:

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

struct node
{
   int data;
   struct node *next;
   struct node *prev;
};

struct node *newnode;
struct node *head = NULL;

void insertfront(int ele)
{
   newnode = (struct node *)malloc(sizeof(struct node));
   if (head == NULL)
   {
      newnode->data = ele;
      newnode->next = NULL;
      newnode->prev = NULL;
      head = newnode;
   }
   else
   {
      newnode->data = ele;
      newnode->next = head;
      head->prev = newnode;
      head = newnode;
   }
}

void insertend(int ele)
{
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = ele;
   newnode->next = NULL;
   if (head != NULL)
   {
      struct node *t;
      t = head;
      while (t->next != NULL)
      {
         t = t->next;
      }
      newnode->prev = t;
      t->next = newnode;
   }
```

```
 else
   {
      newnode->prev = NULL;
      head = newnode;
   }
}

int listsize()
{
   int c = 0;
   struct node *t;
   t = head;
   while (t != NULL)
   {
      c++;
      t = t->next;
   }
   printf("\n The size of the list is %d:\n", c);
   return c;
}

void insertpos(int ele, int pos)
{
   int ls = 0;
   struct node *temp;
   ls = listsize();
   if (head == NULL)
   {
      printf("\nInvalid position to insert a node\n");
      return;
   }
   if (head != NULL && (pos <= 0 || pos > ls))
   {
      printf("\nInvalid position to insert a node\n");
      return;
   }

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

   if (newnode != NULL)
   {
      newnode->data = ele;
      temp = head;
      int count = 1;
      while (count < pos - 1)
      {
         temp = temp->next;
         count++;
      }
      if (pos == 1)
      {
         newnode->next = head;
         head->prev = newnode;
         head = newnode;
      }
```

```
 else
    {
      newnode->next = temp->next;
      if (temp->next != NULL)
        temp->next->prev = newnode;
      temp->next = newnode;
      newnode->prev = temp;
    }
  }
}

void find(int s)
{
  int c = 1;
  struct node *temp;
  temp = head;
  if (head == NULL)
  {
    printf("\n List is empty");
  }
  else
  {
    while (temp->data != s && temp->next != NULL)
    {
      temp = temp->next;
      c++;
    }
    if (temp != NULL && temp->data == s)
    {
      printf("\n Searching ele %d is present in the addr of %p in the pos%d", temp->data,
temp, c);
    }
    else
    {
      printf("\n Searching elem %d is not present", s);
    }
  }
}

void findnext(int s)
{
  struct node *temp;
  temp = head;
  if (temp == NULL || temp->next == NULL)
  {
    printf("No next element ");
  }
  else
  {
    while (temp->data != s)
    {
      temp = temp->next;
    }
    printf("\nNext Element of %d is %d\n", s, temp->next->data);
```

```
 }
}

void findprev(int s)
{
   struct node *temp;
   temp = head;
   if (temp == NULL)
   {
      printf("List is empty ");
   }
   else
   {
      while (temp->data != s)
      {
         temp = temp->next;
      }
      printf("\n The previous ele of %d is %d\n", s, temp->prev->data);
   }
}

void deleteAtBeginning()
{
   if (head == NULL)
   {
      printf("List is empty");
   }
   else
   {
      struct node *t = head;
      head = head->next;
      if (head != NULL)
         head->prev = NULL;
      free(t);
   }
}

void deleteAtEnd()
{
   if (head == NULL)
   {
      printf("\n List is empty");
   }
   else
   {
      struct node *temp = head;
      while (temp->next != NULL)
      {
         temp = temp->next;
      }
      if (temp->prev != NULL)
         temp->prev->next = NULL;
      free(temp);
```

```
 }
}

void delete(int ele)
{
   struct node *t = head;
   if (t->data == ele)
   {
      head = t->next;
      if (head != NULL)
         head->prev = NULL;
      free(t);
   }
   else
   {
      while (t->data != ele)
      {
         t = t->next;
      }
      if (t->next != NULL)
         t->next->prev = t->prev;
      t->prev->next = t->next;
      free(t);
   }
}

void display()
{
   struct node *temp;
   temp = head;
   while (temp != NULL)
   {
      printf("%d-->", temp->data);
      temp = temp->next;
   }
}

int main()
{
   insertfront(10);
   insertfront(20);
   insertfront(30);
   display();
   printf("\n Inserted ele 40 at the end\n");
   insertend(40);
   display();
   insertpos(25, 3);
   display();
   find(25);
   findnext(25);
   findprev(25);
   printf("\n element deleted in the beginning\n");
   deleteAtBeginning();
   display();
```

```
 deleteAtEnd();
  printf("\n Element deleted at the end\n");
  display();
  printf("\n After deleting element 25\n");
  delete(25);
  display();
  return 0;
}
```

OUTPUT:

```
30-->20-->10-->
 Inserted ele 40 at the end
30-->20-->10-->40-->
 The size of the list is 4:
30-->20-->25-->10-->40-->
 Searching ele 25 is present in the addr of 0x55ff3be7d730 in the pos3
Next Element of 25 is 10

 The previous ele of 25 is 20

 element deleted in the beginning
20-->25-->10-->40-->
 Element deleted at the end
20-->25-->10-->
 After deleting element 25
20-->10-->aiml231501167@cselab:~$
```

RESULT: Thus, the program was successfully executed.

Ex. No.:03	Polynomial Manipulation	Date:21/03/2024

Write a C program to implement the following operations on Singly Linked List.

(i) Polynomial Addition

(ii) Polynomial Subtraction

(iii) Polynomial Multiplication

Algorithm:

PROGRAM:

```
aiml2315011167@cselab:~$ gcc program3.c
aiml2315011167@cselab:~$ ./a.out
Enter the values for first polynomial :
Enter the coefficient : 2
Enter the power : 2
Enter 1 to continue : 1
Enter the coefficient : 3
Enter the power : 3
Enter 1 to continue : 0
The polynomial equation is : 2x^2+3x^3
Enter the values for second polynomial :
Enter the coefficient : 2
Enter the power : 2
Enter 1 to continue : 1
Enter the coefficient : 5
Enter the power : 3
Enter 1 to continue : 0
The polynomial equation is : 2x^2+5x^3
The polynomial equation addition result is : 4x^2+8x^3aiml2315011167@cselab:~$
```

RESULT: Thus the program was successfully executed.

Ex. No.:04	Implementation of Stack using Array and Linked List Implementation	Date:21/03/2024

Write a C program to implement a stack using Array and linked List implementation and execute the following operation on stack.

(i) Push an element into a stack

(ii) Pop an element from a stack

(iii) Return the Top most element from a stack

(iv) Display the elements in a stack

Algorithm:

```
1. Push to stack&quot;
2. Pop from Stack:
3. Display data of Stack
4. Exit

Choose Option:1

Enter a value to push into Stack:10

Choose Option:1

Enter a value to push into Stack:20

Choose Option:1

Enter a value to push into Stack:30

Choose Option:1

Enter a value to push into Stack:40

Choose Option:2

 The last element is popped
Choose Option:3

Elements are as:        30
        20
        10


Choose Option:4
aiml231501167@cselab:~$
```

RESULT: Thus the program was successfully executed.

Ex. No.: 05	Infix to Postfix Conversion	Date:28/03/2024

Write a C program to perform infix to postfix conversion using stack.

Algorithm:

```
Enter the infix expression: ab+cd+*ef/*g^

Postfix expression is =>
                abcd+ef/*g*^+aiml231501179@cselab:~$
```

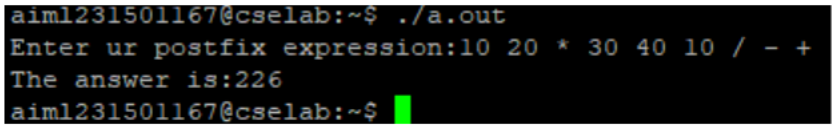
RESULT: Thus, the program was successfully executed.

Ex. No.:06	Evaluating Arithmetic Expression	Date:04/04/2024

Write a C program to evaluate Arithmetic expressions using stack.

Algorithm:

PROGRAM:



RESULT: Thus, the program was successfully executed.

Ex. No.:07	Implementation of Queue using Array and Linked List Implementation	Date:11/04/2024

Write a C program to implement a Queue using Array and linked List implementation and execute the following operation on stack.

(i) Enqueue

(ii) Dequeue

(iii) Display the elements in a Queue

Algorithm:

PROGRAM:

OUTPUT:

```
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 1

Enter data: 2

Element added to the queue.
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 1

Enter data: 3

Element added to the queue.
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 1

Enter data: 4

Element added to the queue.
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 2
Deleted data: 2
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 2
Deleted data: 3
1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 4

1. Add to Queue
2. Delete from Queue
3. Exit
Select an option: 3
aiml231501179@cselab:~$
```

RESULT: Thus, the program was successfully executed.

Ex. No.:08	Tree Traversal	Date:18/04/2024

Write a C program to implement a Binary tree and perform the following tree traversal operation.

(i) Inorder Traversal

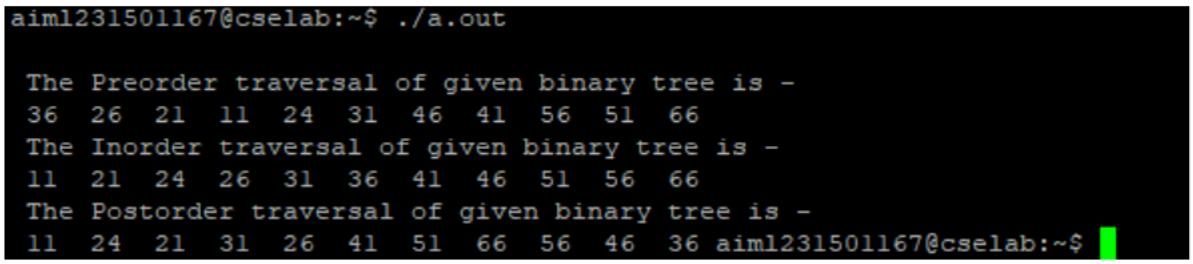
(ii) Preorder Traversal

(iii) Postorder Traversal

Algorithm:

PROGRAM;

OUTPUT:



RESULT: Thus, the program was successfully executed.

Ex. No.:09	Implementation of Binary Search tree	Date:25/04/2024

Write a C program to implement a Binary Search Tree and perform the following operations.

(i) Insert

(ii) Delete

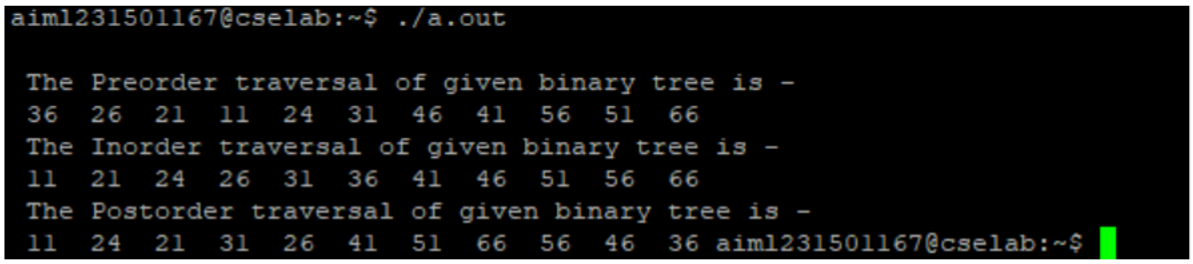
(iii) Search

(iv) Display

Algorithm:

PROGRAM;

OUTPUT:



RESULT: Thus, the program was successfully executed.

Ex. No.: 10	Implementation of AVL Tree	Date:02/05/2024

Write a function in C program to insert a new node with a given value into an AVL tree. Ensure that the tree remains balanced after insertion by performing rotations if necessary. Repeat the above operation to delete a node from AVL tree.

Algorithm:

PROGRAM:

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

struct node
{
 int data;
 struct node* left;
 struct node* right;
 int ht;
};

struct node* root = NULL;

struct node* create(int);
struct node* insert(struct node*, int);
struct node* delete(struct node*, int);
struct node* search(struct node*, int);
struct node* rotate_left(struct node*);
struct node* rotate_right(struct node*);
int balance_factor(struct node*);
int height(struct node*);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);

int main()
{
   int user_choice, data;
```

```
char user_continue = 'y';
struct node* result = NULL;

while (user_continue == 'y' || user_continue == 'Y')
{
   printf("\n\n------- AVL TREE --------\n");
   printf("\n1. Insert");
   printf("\n2. Delete");
   printf("\n3. Search");
   printf("\n4. Inorder");
   printf("\n5. Preorder");
   printf("\n6. Postorder");
   printf("\n7. EXIT");

   printf("\n\nEnter Your Choice: ");
   scanf("%d", &user_choice);

   switch(user_choice)
   {
      case 1:
         printf("\nEnter data: ");
         scanf("%d", &data);
         root = insert(root, data);
         break;

      case 2:
         printf("\nEnter data: ");
         scanf("%d", &data);
         root = delete(root, data);
         break;
```

```
  case 3:
   printf("\nEnter data: ");
   scanf("%d", &data);
   result = search(root, data);
   if (result == NULL)
   {
            printf("\nNode not found!");
   }
   else
   {
      printf("\n Node found");
   }
   break;
case 4:
   inorder(root);
   break;

case 5:
   preorder(root);
   break;

case 6:
   postorder(root);
   break;

case 7:
   printf("\n\tProgram Terminated\n");
   return 1;

default:
```

```
            printf("\n\tInvalid Choice\n");
        }

        printf("\n\nDo you want to continue? ");
        scanf(" %c", &user_continue);
    }

    return 0;
}
struct node* create(int data)
{
    struct node* new_node = (struct node*) malloc (sizeof(struct node));
    if (new_node == NULL)
    {
        printf("\nMemory can&'t be allocated\n");
        return NULL;
    }
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}
struct node* rotate_left(struct node* root)
{
    struct node* right_child = root->right;
    root->right = right_child->left;
    right_child->left = root;
    root->ht = height(root);
    right_child->ht = height(right_child);
```

```
            return right_child;
}
struct node* rotate_right(struct node* root)
{
   struct node* left_child = root->left;
   root->left = left_child->right;
   left_child->right = root;

   root->ht = height(root);
   left_child->ht = height(left_child);
         return left_child;
}

int balance_factor(struct node* root)
{
   int lh, rh;
   if (root == NULL)
      return 0;
   if (root->left == NULL)
      lh = 0;
   else
      lh = 1 + root->left->ht;
   if (root->right == NULL)
      rh = 0;
   else
      rh = 1 + root->right->ht;
   return lh - rh;
}
int height(struct node* root)
 {
   int lh, rh;
```

```
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;

    if (lh > rh)
        return (lh);
    return (rh);
}

struct node* insert(struct node* root, int data)
{
    if (root == NULL)
    {
        struct node* new_node = create(data);
        if (new_node == NULL)
        {
            return NULL;
        }
        root = new_node;
    }
    else if (data > root->data)
    {
            root->right = insert(root->right, data);
```

```
    if (balance_factor(root) == -2)
    {
        if (data > root->right->data)
        {
            root = rotate_left(root);
        }
        else
        {
            root->right = rotate_right(root->right);
            root = rotate_left(root);
        }
    }
}
else
{
    root->left = insert(root->left, data);
      if (balance_factor(root) == 2)
    {
        if (data < root->left->data)
        {
            root = rotate_right(root);
        }
        else
        {
            root->left = rotate_left(root->left);
            root = rotate_right(root);
        }
    }
}
root->ht = height(root);
return root;
```

```
}

struct node * delete(struct node *root, int x)
{
   struct node * temp = NULL;

   if (root == NULL)
   {
      return NULL;
   }

   if (x > root->data)
   {
      root->right = delete(root->right, x);
      if (balance_factor(root) == 2)
      {
         if (balance_factor(root->left) >= 0)
         {
            root = rotate_right(root);
         }
         else
         {
            root->left = rotate_left(root->left);
            root = rotate_right(root);
         }
      }
   }
   else if (x < root->data)
   {
      root->left = delete(root->left, x);
```

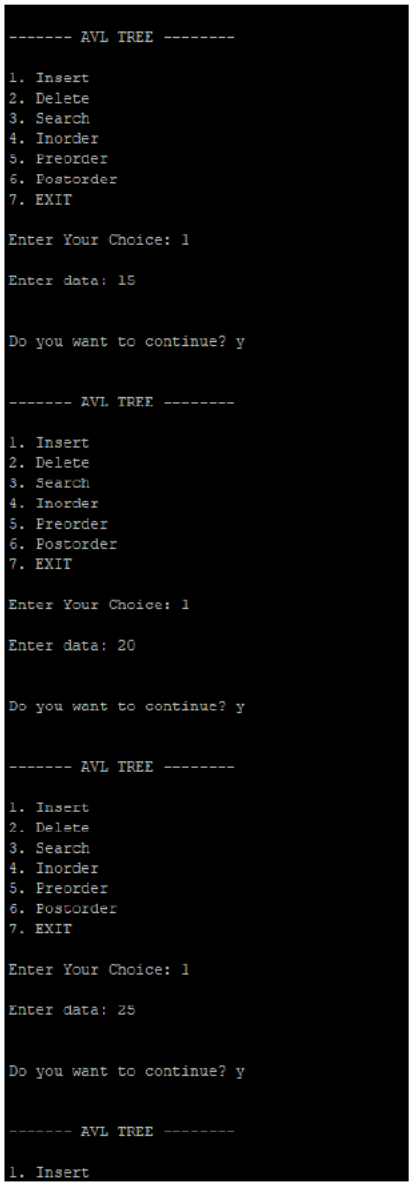
```
            if (balance_factor(root) == -2)
      {
         if (balance_factor(root->right) <= 0)
         {
            root = rotate_left(root);
         }
         else
{
root->right = rotate_right(root->right);
root = rotate_left(root);
}
}
}
else
{
if (root->right != NULL)
{
temp = root->right;
while (temp->left != NULL)
temp = temp->left;

root->data = temp->data;
root->right = delete(root->right, temp->data);
if (balance_factor(root) == 2)
{
if (balance_factor(root->left) >= 0)
{
root = rotate_right(root);
}
```

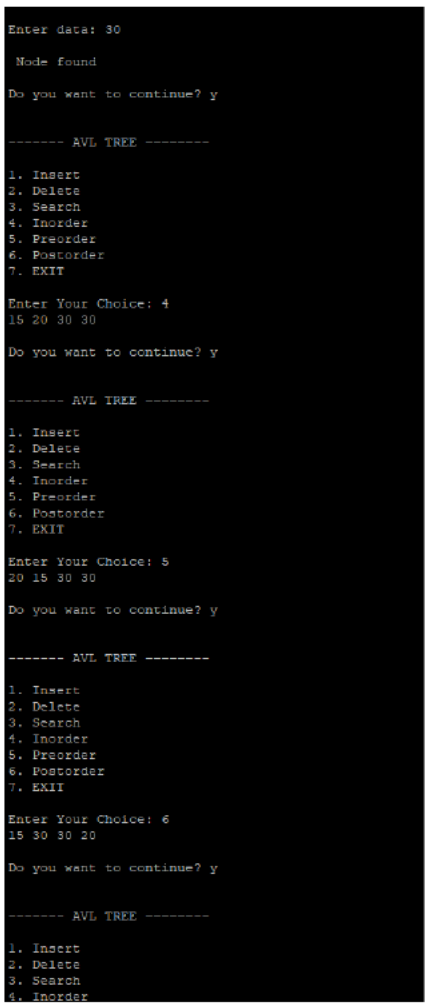
```
else
{
root->left = rotate_left(root->left);
root = rotate_right(root);
}
}
}
else
{
return (root->left);
}
}
root->ht = height(root);
return (root);
}
struct node* search(struct node* root, int key)
{
 if(root == NULL)
{
return NULL;
}

if(root->data == key)
{
return root;
}
if(key > root->data)
{
search(root->right, key);
}
```

```
else
{
      search(root->left, key);
}
}
void inorder(struct node* root)
{
if (root == NULL)
{
 return;
}
inorder(root->left);
printf("%d ", root->data);
inorder(root->right);
}
void preorder(struct node* root)
{
   if(root == NULL)
{
return;
}
printf("%d ", root->data);
preorder(root->left);
preorder(root->right);
}
void postorder(struct node* root)
{
if(root == NULL)
{
   return;
}
```

```
postorder(root->left);
postorder(root->right);
printf("%d ", root->data);
}
```

OUTPUT:





RESULT: Thus, the program was successfully executed.

Ex. No.: 11	Topological Sorting	Date:09/05/2024

Write a C program to create a graph and display the ordering of vertices.

Algorithm:

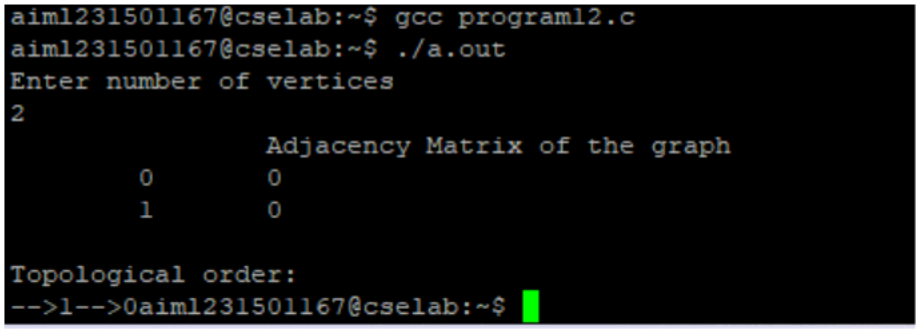
PROGRAM:

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

int s[100], j, res[100;
void AdjacencyMatrix(int a[][100], int n) {
 int i, j;
for (i = 0; i < n; i++) {
for (j = 0; j <= n; j++) {
a[i][j] = 0;
}
}
for (i = 1; i < n; i++) {
for (j = 0; j < i; j++) {
a[i][j] = rand() % 2;
a[j][i] = 0;
}
}
}

void dfs(int u, int n, int a[][100]) {
int v;
s[u] = 1;
for (v = 0; v < n - 1; v++) {
if (a[u][v] == 1 && s[v] == 0) {
dfs(v, n, a);

}
}
j += 1;
```

```
res[j] = u;
}

void topological_order(int n, int a[][100]) {
int i, u;
for (i = 0; i < n; i++) {
s[i] = 0;
}
j = 0;
for (u = 0; u < n; u++) {
if (s[u] == 0) {
dfs(u, n, a);
}
}
return;
}
int main() {
int a[100][100], n, i, j;

printf("Enter number of vertices\n");
scanf("%d", &n);

AdjacencyMatrix(a, n);
printf("\t\tAdjacency Matrix of the graph\n"); /* PRINT ADJACENCY MATRIX */

for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
printf("\t%d", a[i][j]);
}
printf("\n");
}
```

```
printf("\nTopological order:\n");

topological_order(n, a);

for (i = n; i >= 1; i--) {
printf("-->%d", res[i]);
}
return 0;
}
```

OUTPUT:



RESULT: Thus, the program was successfully executed.

Ex. No.:12	Graph Traversal	Date:09/05/2024

Write a C program to create a graph and perform a Breadth First Search and Depth First Search.

Algorithm:

PROGRAM: BFS

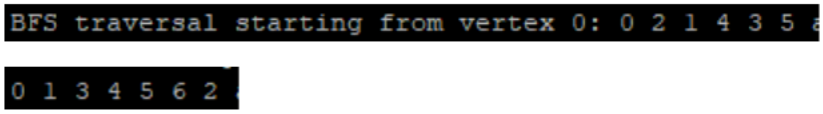
```
#include <stdio.h>
#include <stdlib.h>

struct node {
int vertex;
struct node* next;

};
struct adj_list {
struct node* head;
};

struct graph {
int num_vertices;
struct adj_list* adj_lists;
int* visited;
};
struct node* new_node(int vertex) {
struct node* new_node = (struct node*)malloc(sizeof(struct node));
new_node->vertex = vertex;
new_node->next = NULL;
return new_node;
}
struct graph* create_graph(int n) {
struct graph* graph = (struct graph*)malloc(sizeof(struct graph));
graph->num_vertices = n;
graph->adj_lists = (struct adj_list*)malloc(n * sizeof(struct adj_list));
graph->visited = (int*)malloc(n * sizeof(int));
```

```
int i;
for (i = 0; i< n; i++) {
graph->adj_lists[i].head = NULL;
graph->visited[i] = 0;
}

return graph;
}
void add_edge(struct graph* graph, int src, int dest) {
struct node* new_node1 = new_node(dest);
new_node1->next = graph->adj_lists[src].head;
graph->adj_lists[src].head = new_node1;
struct node* new_node2 = new_node(src);
new_node2->next = graph->adj_lists[dest].head;
graph->adj_lists[dest].head = new_node2;
}
void bfs(struct graph* graph, int v) {
int queue[1000];
int front = -1;
int rear = -1;
graph->visited[v] = 1;
queue[++rear] = v;
while (front != rear) {
int current_vertex = queue[++front];
printf("%d ", current_vertex);
struct node* temp = graph->adj_lists[current_vertex].head;
while (temp != NULL) {
int adj_vertex = temp->vertex;
```

```
if (graph->visited[adj_vertex] == 0) {
graph->visited[adj_vertex] = 1;
queue[++rear] = adj_vertex;
}
temp = temp->next;
}
}
}

int main() {
struct graph* graph = create_graph(6);
add_edge(graph, 0, 1);
add_edge(graph, 0, 2);
add_edge(graph, 1, 3);
add_edge(graph, 1, 4);
add_edge(graph, 2, 4);
add_edge(graph, 3, 4);
add_edge(graph, 3, 5);
add_edge(graph, 4,5);
printf("BFS traversal starting from vertex 0: ");
bfs(graph, 0);

return 0;
}
```

DFS:

OUTPUT:



RESULT: Thus, the program was successfully executed.

Ex. No.:13	Graph Traversal	Date:16/05/2014

Write a C program to create a graph and find a minimum spanning tree using prim's algorithm.

Algorithm:

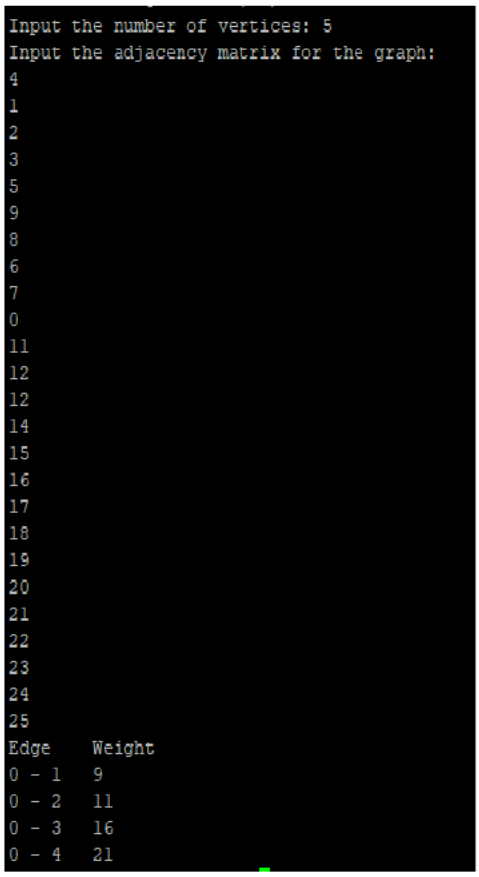
PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int vertex;
struct node* next;

};
struct adj_list {
struct node* head;
};
struct graph {
int num_vertices;
struct adj_list* adj_lists;
int* visited;
};
struct node* new_node(int vertex) {
struct node* new_node = (struct node*)malloc(sizeof(struct node));
new_node->vertex = vertex;
new_node->next = NULL;
return new_node;
}
struct graph* create_graph(int n) {
struct graph* graph = (struct graph*)malloc(sizeof(struct graph));
graph->num_vertices = n;
graph->adj_lists = (struct adj_list*)malloc(n * sizeof(struct adj_list));
graph->visited = (int*)malloc(n * sizeof(int));

int i;
for (i = 0; i< n; i++) {
```

```
graph->adj_lists[i].head = NULL;
graph->visited[i] = 0;
}

return graph;
}

void add_edge(struct graph* graph, int src, int dest) {
struct node* new_node1 = new_node(dest);
new_node1->next = graph->adj_lists[src].head;
graph->adj_lists[src].head = new_node1;
struct node* new_node2 = new_node(src);
new_node2->next = graph->adj_lists[dest].head;
graph->adj_lists[dest].head = new_node2;
}
void bfs(struct graph* graph, int v) {
int queue[1000];
int front = -1;
int rear = -1;
graph->visited[v] = 1;
queue[++rear] = v;
while (front != rear) {
int current_vertex = queue[++front];
printf("%d ", current_vertex);
struct node* temp = graph->adj_lists[current_vertex].head;
while (temp != NULL) {
int adj_vertex = temp->vertex;
if (graph->visited[adj_vertex] == 0) {
graph->visited[adj_vertex] = 1;
queue[++rear] = adj_vertex;
}
```

```
temp = temp->next;
}
}
}

int main() {
struct graph* graph = create_graph(6);
add_edge(graph, 0, 1);
add_edge(graph, 0, 2);
add_edge(graph, 1, 3);
add_edge(graph, 1, 4);
add_edge(graph, 2, 4);
add_edge(graph, 3, 4);
add_edge(graph, 3, 5);
add_edge(graph, 4,5);
printf("BFS traversal starting from vertex 0: ");
bfs(graph, 0);

return 0;
}
```

OUTPUT:



RESULT: Thus, the program was successfully executed.

Ex. No.:14	Graph Traversal	Date: 16/05/2024

Write a C program to create a graph and find the shortest path using Dijkstra's Algorithm.

Algorithm:

PROGRAM;

```
#include <stdio.h>
#include <limits.h>
#define MAX_VERTICES 100
int minDistance(int dist[], int sptSet[], int vertices) {
int min = INT_MAX, minIndex;
for (int v = 0; v < vertices; v++) {
if (!sptSet[v] && dist[v] < min) {
min = dist[v];
minIndex = v;
}
}
return minIndex;
}
void printSolution(int dist[], int vertices) {
printf("Vertex \tDistance from Source\n");
for (int i = 0; i < vertices; i++) {
printf("%d \t%d\n", i, dist[i]);
}
}
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int
vertices) {
int dist[MAX_VERTICES];
int sptSet[MAX_VERTICES];
for (int i = 0; i < vertices; i++) {
dist[i] = INT_MAX;
sptSet[i] = 0;
}
dist[src] = 0;
```

```
for (int count = 0; count < vertices - 1; count++) {
int u = minDistance(dist, sptSet, vertices);
sptSet[u] = 1;

for (int v = 0; v < vertices; v++) {
if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX &&
dist[u] + graph[u][v] < dist[v]) {
dist[v] = dist[u] + graph[u][v];
}
}
}
printSolution(dist, vertices);
}
int main() {
int vertices;
printf("Input the number of vertices: ");
scanf("%d", &vertices);
if (vertices <= 0 || vertices > MAX_VERTICES) {
printf("Invalid number of vertices. Exiting...\n");
return 1;
}
int graph[MAX_VERTICES][MAX_VERTICES];
printf("Input the adjacency matrix for the graph (use INT_MAX forinfinity):\n");
for (int i = 0; i < vertices; i++) {

for (int j = 0; j < vertices; j++) {
scanf("%d", &graph[i][j]);
}
}
int source;
```

```
printf("Input the source vertex: ");
scanf("%d", &source);
if (source < 0 || source >= vertices) {
printf("Invalid source vertex. Exiting...\n");
return 1;
}
dijkstra(graph, source, vertices);
return 0;
}
```

OUTPUT:

```
Input the number of vertices: 3
Input the adjacency matrix for the graph (use INT_MAX forinfinity):
1
2
3
4
5
6
7
8
9
Input the source vertex: 1
Vertex  Distance from Source
0       4
1       0
2       6
```

RESULT: Thus, the program was successfully executed.

Ex. No.:15	Sorting	Date:23/05/2024

Write a C program to take n numbers and sort the numbers in ascending order. Try to implement the same using the following sorting techniques.

1. **Quick Sort**
2. **Merge Sort**

Algorithm:

PROGRAM:

QUICK SORT:

OUTPUT:

```
Original array:19171512161841113
Sorted array:41112131516171819aim
```

RESULT: Thus, the program was successfully executed.

Ex. No.: 16	Hashing	Date:30/05/2024

Write a C program to create a hash table and perform collision resolution using the following techniques.

(i) Open addressing

(ii) Closed Addressing

(iii) Rehashing

Algorithm:

PROGRAM:

A. OPEN ADDRESSING:

B. CLOSED ADDRESSING;

C. REHASHING:

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

typedef struct Node {
int key;
int value;
struct Node* next;
} Node;

typedef struct HashTable {
int size;
int count;
Node** table;
} HashTable;

Node* createNode(int key, int value) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->key = key;
newNode->value = value;
newNode->next = NULL;
return newNode;
}

HashTable* createTable(int size) {
HashTable* newTable = (HashTable*)malloc(sizeof(HashTable));
newTable->size = size;
newTable->count = 0;
newTable->table = (Node**)malloc(sizeof(Node*) * size);
for (int i = 0; i < size; i++) {

newTable->table[i] = NULL;
}
return newTable;
}

int hashFunction(int key, int size) {
return key % size;
}

void insert(HashTable* hashTable, int key, int value);

void rehash(HashTable* hashTable) {
int oldSize = hashTable->size;
Node** oldTable = hashTable->table;

// New size is typically a prime number or double the old size
int newSize = oldSize * 2;
hashTable->table = (Node**)malloc(sizeof(Node*) * newSize);
hashTable->size = newSize;
hashTable->count = 0;
for (int i = 0; i < newSize; i++) {
hashTable->table[i] = NULL;
}
```

```
for (int i = 0; i < oldSize; i++) {
Node* current = oldTable[i];
while (current != NULL) {
insert(hashTable, current->key, current->value);
Node* temp = current;

current = current->next;
free(temp);
}
}

free(oldTable);
}

void insert(HashTable* hashTable, int key, int value) {
if ((float)hashTable->count / hashTable->size >= 0.75) {
rehash(hashTable);
}

int hashIndex = hashFunction(key, hashTable->size);
Node* newNode = createNode(key, value);
newNode->next = hashTable->table[hashIndex];
hashTable->table[hashIndex] = newNode;
hashTable->count++;
}

int search(HashTable* hashTable, int key) {
int hashIndex = hashFunction(key, hashTable->size);
Node* current = hashTable->table[hashIndex];
while (current != NULL) {
if (current->key == key) {
return current->value;
}
current = current->next;
}

return -1;
}

void delete(HashTable* hashTable, int key) {
int hashIndex = hashFunction(key, hashTable->size);
Node* current = hashTable->table[hashIndex];
Node* prev = NULL;
while (current != NULL && current->key != key) {
prev = current;
current = current->next;
}
if (current == NULL) {
return;
}
if (prev == NULL) {
hashTable->table[hashIndex] = current->next;
} else {
prev->next = current->next;
```

```
}
free(current);
hashTable->count--;
}

void freeTable(HashTable* hashTable) {
for (int i = 0; i < hashTable->size; i++) {
Node* current = hashTable->table[i];
while (current != NULL) {
Node* temp = current;
current = current->next;

free(temp);
}
}
free(hashTable->table);
free(hashTable);
}

int main() {
HashTable* hashTable = createTable(5);

insert(hashTable, 1, 10);
insert(hashTable, 2, 20);
insert(hashTable, 3, 30);
insert(hashTable, 4, 40);
insert(hashTable, 5, 50);
insert(hashTable, 6, 60); // This should trigger rehashing

printf("Value for key 1: %d\n", search(hashTable, 1));
printf("Value for key 2: %d\n", search(hashTable, 2));
printf("Value for key 3: %d\n", search(hashTable, 3));
printf("Value for key 4: %d\n", search(hashTable, 4));
printf("Value for key 5: %d\n", search(hashTable, 5));
printf("Value for key 6: %d\n", search(hashTable, 6));

delete(hashTable, 3);
printf("Value for key 3 after deletion: %d\n", search(hashTable, 3));

freeTable(hashTable);
return 0;

}
```

OUTPUT:

```
aiml231501167@cselab:~$ gcc program16C.c
aiml231501167@cselab:~$ ./a.out
Value for key 1: 10
Value for key 2: 20
Value for key 3: 30
Value for key 4: 40
Value for key 5: 50
Value for key 6: 60
Value for key 3 after deletion: -1
aiml231501167@cselab:~$
```

```
Value for key 1: 10
Value for key 2: 20
Value for key 12: 30
Value for key 3: -1
Value for key 2 after deletion: -1
aiml231501167@cselab:~$
```

```
aiml231501167@cselab:~$ gcc program16A.c
aiml231501167@cselab:~$ ./a.out
List before sorting
10 14 19 26 27 31 33 35 42 44 0
List after sorting
0 10 14 19 26 27 31 33 35 42 44 aiml231501167@cselab:~$
```

RESULT: Thus, the program was successfully executed.

Rajalakshmi Institutions

Rajalakshmi Engineering College

Rajalakshmi Nagar Thandalam, Chennai - 602 105.

Phone : +91-44-67181111, 67181112

Website : www.rajalakshmi.org