**Hash Tables**

**LAB # 11**



**Data Structures & Algorithms**

Submitted by: **Shah Raza**

Registration No: **18PWCSE1658**

Class Section: **B**

“On my honor, as a student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to: **Dr. Khurram Shehzad Khattak**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab Objectives:**

Objectives of this lab are as follows:

* Implementing and performing basic operations on Hash Table data structure.

**Task # 1:**

Implement a Hash table with the mentioned features.

**Code:**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*next\_node;

};

class LinkedList

{

public:

Node \*head;

LinkedList()

{

head=NULL;

}

void InsertAtStart(int data)

{

Node \*temp,\*p;

temp= new Node;

temp->data=data;

if(head==NULL)

{

head=temp;

head->next\_node=NULL;

}

else

{

p=head;

head=temp;

head->next\_node=p;

}

cout<<"Element inserted at the start.\n";

}

void InsertAtEnd(int data)

{

Node \*temp,\*p;

temp= new Node;

temp->data=data;

if(head==NULL)

{

head=temp;

head->next\_node=NULL;

return;

}

p=head;

while(p->next\_node!=NULL)

{

p=p->next\_node;

}

temp->next\_node=NULL;

p->next\_node=temp;

cout<<"Element inserted at the last.\n";

}

void Insert(int data, int pos)

{

int lastpos=0;

Node \*temp,\*p,\*s;

temp= new Node;

temp->data=data;

p=head;

while(p!=NULL)

{

p=p->next\_node;

lastpos++;

}

if(pos==1)

{

InsertAtStart(data);

}

else if(pos>1 && pos<=lastpos)

{

p=head;

for(int i=1;i<pos;i++)

{

s=p;

p=p->next\_node;

}

s->next\_node=temp;

temp->next\_node=p;

}

else

cout<<"Position out of range.\n";

}

void DeleteFromStart()

{

if(head==NULL)

cout<<"List is empty.\n";

else

{

Node \*s;

s=head;

head=head->next\_node;

delete s;

cout<<"Element Deleted.\n";

}

}

void DeleteFromEnd()

{

int counter=0;

Node \*s;

s=head;

while(s!=NULL)

{

s=s->next\_node;

counter++;

}

if(counter==1)

{

head=NULL;

}

else

{

s=head;

while(s->next\_node->next\_node!=NULL)

{

s=s->next\_node;

}

delete (s->next\_node);

s->next\_node=NULL;

}

cout<<"Element Deleted.\n";

}

void DeleteAtPosition(int pos)

{

int counter=0;

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

if(pos==1)

{

DeleteFromStart();

return;

}

Node \*s,\*p;

s=head;

while(s!=NULL)

{

s=s->next\_node;

counter++;

}

if(pos>1 && pos<=counter)

{

s=head;

for(int i=1;i<pos;i++)

{

p=s;

s=s->next\_node;

}

p->next\_node=s->next\_node;

delete s;

cout<<"Element Deleted.\n";

}

else

cout<<"Position out of Range.\n";

}

int Search(int data)

{

if(head==NULL)

return -1;

int pos=0;

Node \*s;

s=head;

while(s!=NULL)

{

pos++;

if(s->data==data)

return pos;

s=s->next\_node;

}

return -1;

}

void Print()

{

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

Node \*s;

s=head;

cout<<"Elements of the list are: ";

while(s!=NULL)

{

cout<<s->data<<"-->";

s=s->next\_node;

}

cout<<"NULL\n";

}

};

class Hash

{

int Max=5;

LinkedList \*table=new LinkedList[Max];

public:

int hashFunction(int x)

{

return (x%Max);

}

void insertItem(int d)

{

int index=hashFunction(d);

table[index].InsertAtEnd(d);

}

void DeleteItem(int d)

{

int index = hashFunction(d);

int pos= table[index].Search(d);

if(pos>=0)

table[index].DeleteAtPosition(pos);

else

cout<<"Element not found\n";

}

void Display()

{

for(int i=0;i<Max;i++)

{

cout<<i<<" --> ";

table[i].Print();

}

}

};

int main()

{

int choice,key;

Hash Hsh;

do

{

cout << "Enter choice: " << endl;

cout << "1.Insert " <<endl;

cout << "2.Remove " <<endl;

cout << "3.display " << endl;

cout << "4.Exit" << endl;

cin >> choice;

switch(choice)

{

case 1:

cout << "Enter key : " << endl;

cin >> key ;

Hsh.insertItem(key);

break;

case 2:

cout << "Enter key: " << endl;

cin >> key;

Hsh.DeleteItem(key);

break;

case 3:

Hsh.Display();

break;

}

}while(choice!=4);

return 0;

}

**Output:**