Neh Samir Joshi

19BEC1112

11/02/2022

**CSE2003**

**Data Structures and Algorithms**

**[LAB]**

**LAB – 5**

**Heaps and Hashing**

**Aim:** To implement min-heaps and max-heaps and various types of Hashing using C.

**Software Required:** Code editor (e.g. VS Code, Dev C++), GCC/G++ compiler

**Task 1:** To create a min-heap and perform basic functions.

**Code:**

#include<stdio.h>

#define max 50

int heap[max]; *// The Heap*

int n; *//Number of elements*

void heapify();

void maxheapify(int i);

void insert(int var);

void delete(int var);

int main() {

    printf("\nEnter the number of elements to be inserted : ");

    scanf("%d",&n);

    printf("\nEnter the elements of the heap : ");

    int i;

    for(i=0;i<n;i++)

        scanf("%d",&heap[i]);

    heapify();

    while(1){

        int c;

        printf("\n1.Insert\n2.Delete\n3.Exit\nEnter your Choice : ");

        scanf("%d",&c);

        int ele=0;

        switch(c){

            case 1:{

                printf("\nEnter the element to be inserted : ");

                scanf("%d",&ele);

                insert(ele);

                break;

            }

            case 2:{

                printf("\nEnter the element to be deleted : ");

                scanf("%d",&ele);

                delete(ele);

                break;

            }

            case 3: return 0;

        }

        heapify();

        printf("\nHeap is :\n");

        for(i=0;i<n;i++)

            printf("\n%d",heap[i]);

    }

    return 0;

}

void minheapify(int i){

    int small = i;

    int l = 2\*i + 1;

    int r = 2\*i + 2;

    if (l < n && heap[l] < heap[small])

        small = l;

    if (r < n && heap[r] < heap[small])

        small = r;

    if (small != i){

        int tmp = heap[i];

        heap[i]=heap[small];

        heap[small]=tmp;

        minheapify(small);

    }

}

void heapify(){

    int i;

    for (int i = n / 2 - 1; i >= 0; i--)

        minheapify(i);

}

void insert(int var){

    heap[n++] = var;

    return;

}

void delete(int var){

    int i;

    for(i=0;i<n;i++)

        if(heap[i]==var)

            break;

    if(i==n)

        printf("\nElement %d not found",var);

    else{

        heap[i] = heap[n-1];

        heap[n-1] = '\0';

        printf("\nElement %d deleted",var);

    }

}

**Output:**



**Task 2:** To create a max-heap and perform basic operations on them.

**Code:**

#include<stdio.h>

#define max 50

int heap[max]; *// The Heap*

int n; *//Number of elements*

void heapify();

void maxheapify(int i);

void insert(int var);

void delete(int var);

int main() {

    printf("\nEnter the number of elements to be inserted : ");

    scanf("%d",&n);

    printf("\nEnter the elements of the heap : ");

    int i;

    for(i=0;i<n;i++)

        scanf("%d",&heap[i]);

    heapify();

    while(1){

        int c;

        printf("\n1.Insert\n2.Delete\n3.Exit\nEnter your Choice : ");

        scanf("%d",&c);

        int ele=0;

        switch(c){

            case 1:{

                printf("\nEnter the element to be inserted : ");

                scanf("%d",&ele);

                insert(ele);

                break;

            }

            case 2:{

                printf("\nEnter the element to be deleted : ");

                scanf("%d",&ele);

                delete(ele);

                break;

            }

            case 3: return 0;

        }

        heapify();

        printf("\nHeap is :\n");

        for(i=0;i<n;i++)

            printf("\n%d",heap[i]);

    }

    return 0;

}

void maxheapify(int i){

    int largest = i;

    int l = 2\*i + 1;

    int r = 2\*i + 2;

    if (l < n && heap[l] > heap[largest])

        largest = l;

    if (r < n && heap[r] > heap[largest])

        largest = r;

    if (largest != i){

        int tmp = heap[i];

        heap[i]=heap[largest];

        heap[largest]=tmp;

        maxheapify(largest);

    }

}

void heapify(){

    int i;

    for (int i = n / 2 - 1; i >= 0; i--)

        maxheapify(i);

}

void insert(int var){

    heap[n++] = var;

    return;

}

void delete(int var){

    int i;

    for(i=0;i<n;i++)

        if(heap[i]==var)

            break;

    if(i==n)

        printf("\nElement %d not found",var);

    else{

        heap[i] = heap[n-1];

        heap[n-1] = '\0';

        printf("\nElement %d deleted",var);

    }

}

**Output:**



**Task 3:** To implement basic hashing-chaining using C.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 15

struct node

{

    int data;

    struct node \*next;

};

struct node \*head[TABLE\_SIZE] = {NULL}, \*c;

void insert()

{

    int i, key;

    printf("\nenter a value to insert into hash table\n");

    scanf("%d", &key);

    i = key % TABLE\_SIZE;

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

    newnode->data = key;

    newnode->next = NULL;

    if (head[i] == NULL)

        head[i] = newnode;

    else

    {

        c = head[i];

        while (c->next != NULL)

        {

            c = c->next;

        }

        c->next = newnode;

    }

}

void search()

{

    int key, index;

    printf("\nenter the element to be searched\n");

    scanf("%d", &key);

    index = key % TABLE\_SIZE;

    if (head[index] == NULL)

        printf("\n Search element not found\n");

    else

    {

        for (c = head[index]; c != NULL; c = c->next)

        {

            if (c->data == key)

            {

                printf("search element found\n");

                break;

            }

        }

        if (c == NULL)

            printf("\n Search element not found\n");

    }

}

void display()

{

    int i;

    for (i = 0; i < TABLE\_SIZE; i++)

    {

        printf("\Index %d: ", i);

        if (head[i] == NULL)

        {

            printf("No Hash Entry\n");

        }

        else

        {

            for (c = head[i]; c != NULL; c = c->next){

                printf("%d->", c->data);}

                printf("\n");

        }

    }

}

int main()

{

    int opt, key, i;

    while (1)

    {

        printf("\nPress \n1. Enter a key(chaining)\n2. Display \n3. Search \n4.Exit \n");

        scanf("%d", &opt);

        switch (opt)

        {

        case 1:

            insert();

            break;

        case 2:

            display();

            break;

        case 3:

            search();

            break;

        case 4:

            exit(0);

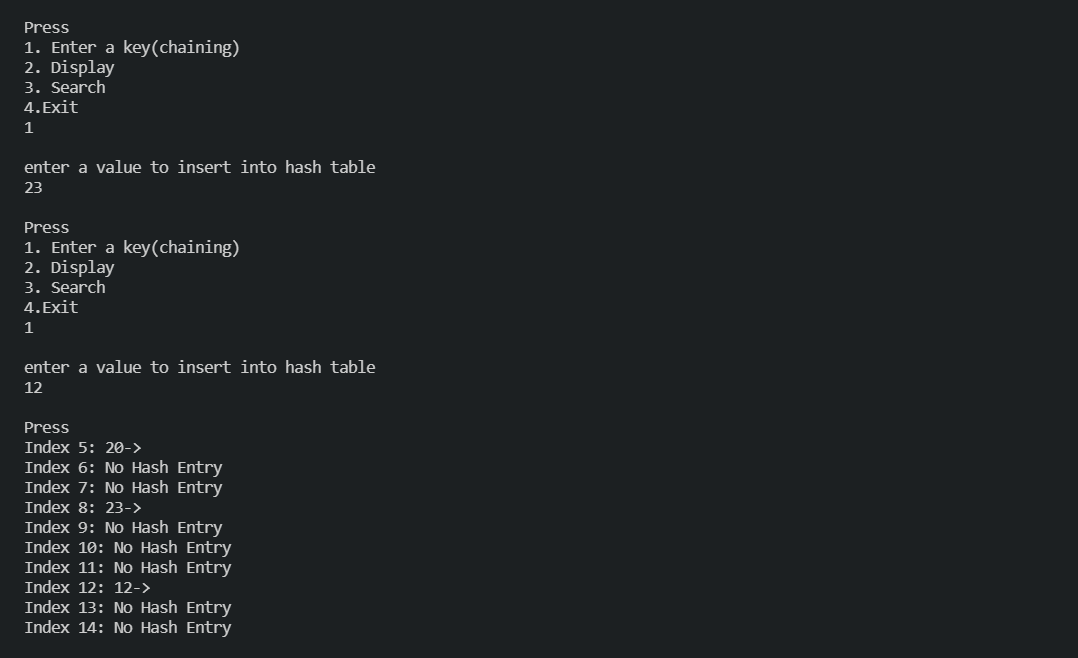
        }

    }

}

*//Post-lab Quadratic Probing and Double Hashing*

**Output:**

****

**Task 4:** To learn about linear probing and implement it.

**Code:**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

 int key,index,i,flag=0,hkey;

 printf("\nenter a value to insert into hash table\n");

 scanf("%d",&key);

 hkey=key%TABLE\_SIZE;

 for(i=0;i<TABLE\_SIZE;i++)

    {

     index=(hkey+i)%TABLE\_SIZE;

     if(h[index] == NULL)

     {

        h[index]=key;

         break;

     }

    }

    if(i == TABLE\_SIZE)

     printf("\nelement cannot be inserted\n");

}

void search()

{

 int key,index,i,flag=0,hkey;

 printf("\nenter search element\n");

 scanf("%d",&key);

 hkey=key%TABLE\_SIZE;

 for(i=0;i<TABLE\_SIZE; i++)

 {

    index=(hkey+i)%TABLE\_SIZE;

    if(h[index]==key)

    {

      printf("value is found at index %d",index);

      break;

    }

  }

  if(i == TABLE\_SIZE)

    printf("\n value is not found\n");

}

void display()

{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE\_SIZE; i++)

  printf("\nat index %d \t value =  %d",i,h[i]);

}

int main()

{

    int opt,i;

    while(1)

    {

        printf("\nPress \n1. Enter a key(linear probing)\n2. Display \n3. Search \n4.Exit \n");

        scanf("%d",&opt);

        switch(opt)

        {

            case 1:

                insert();

                break;

            case 2:

                display();

                break;

            case 3:

                search();

                break;

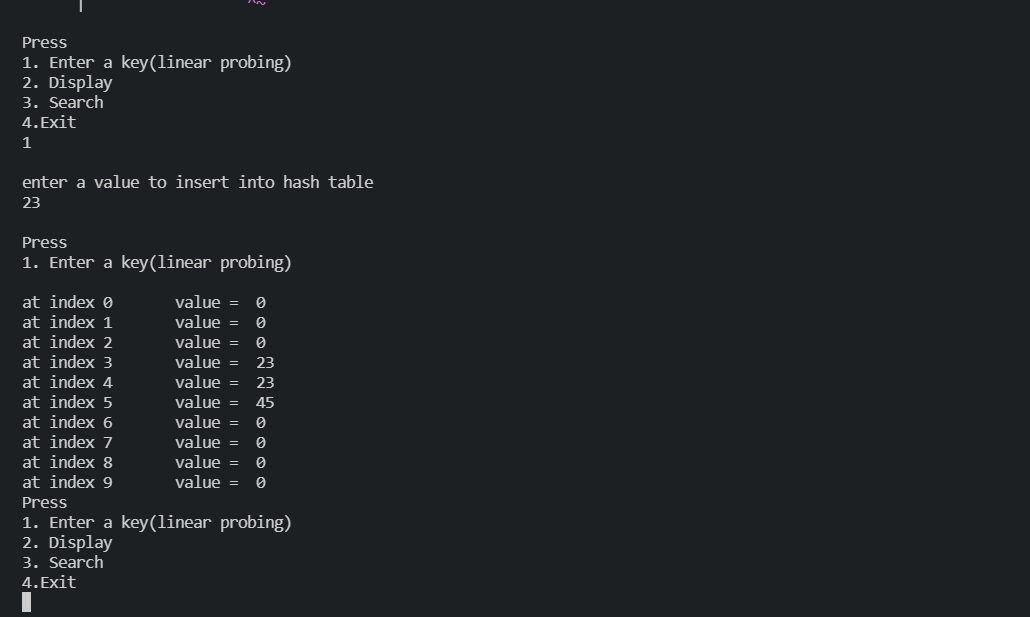
            case 4:exit(0);

        }

    }

}

**Output:**



**Task 5:** To implement quadratic probing and visualize it using C.

**Code:**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

 int key,index,i,flag=0,hkey;

 printf("\nenter a value to insert into hash table\n");

 scanf("%d",&key);

 hkey=key%TABLE\_SIZE;

 for(i=0;i<TABLE\_SIZE;i++)

{

     index=(hkey+i\*i)%TABLE\_SIZE;

     if(h[index] == NULL)

     {

        h[index]=key;

         break;

     }

}

if(i == TABLE\_SIZE)

     printf("\nelement cannot be inserted\n");

}

void search()

{

 int key,index,i,flag=0,hkey;

 printf("\nenter search element\n");

 scanf("%d",&key);

 hkey=key%TABLE\_SIZE;

 for(i=0;i<TABLE\_SIZE; i++)

 {

    index=(hkey+i\*i)%TABLE\_SIZE;

    if(h[index]==key)

    {

      printf("value is found at index %d",index);

      break;

    }

  }

  if(i == TABLE\_SIZE)

    printf("\n value is not found\n");

}

void display()

{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE\_SIZE; i++)

              printf("\nat index %d \t value =  %d",i,h[i]);

}

int main()

{

    int opt,i;

    while(1)

    {

        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

        scanf("%d",&opt);

        switch(opt)

        {

            case 1:

                insert();

                break;

            case 2:

                display();

                break;

            case 3:

                search();

                break;

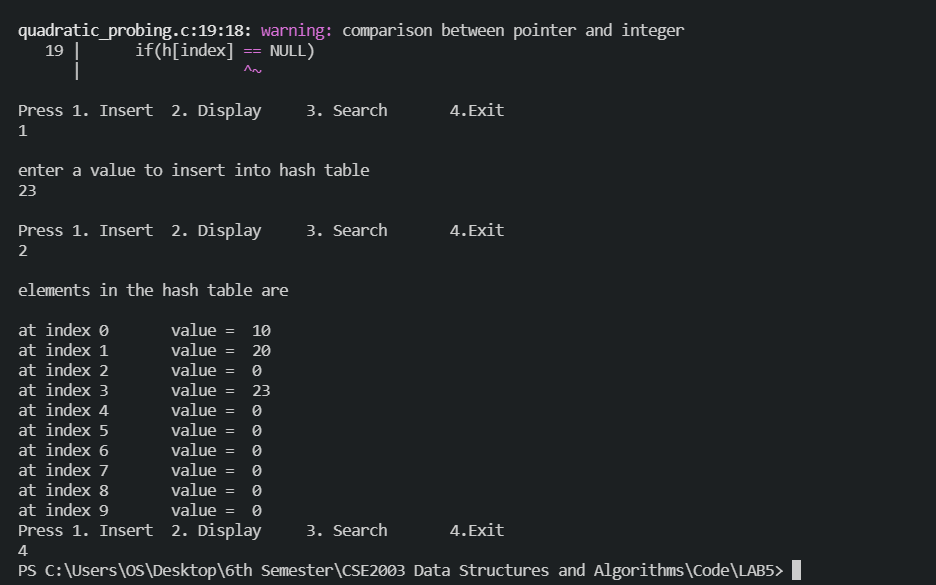
            case 4:exit(0);

        }

    }

}

**Output:**



**Task 6:** To learn and implement double hashing in C.

**Code:**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

 int key,index,i,flag=0,hkey,hash2;

 printf("\nenter a value to insert into hash table\n");

 scanf("%d",&key);

 hkey=key%TABLE\_SIZE;

 hash2 = 7-(key %7);

 for(i=0;i<TABLE\_SIZE;i++)

 {

    index=(hkey+i\*hash2)%TABLE\_SIZE;

    if(h[index] == NULL)

    {

        h[index]=key;

        break;

    }

 }

 if(i == TABLE\_SIZE)

     printf("\nelement cannot be inserted\n");

}

void search()

{

 int key,index,i,flag=0,hash2,hkey;

 printf("\nenter search element\n");

 scanf("%d",&key);

  hkey=key%TABLE\_SIZE;

 hash2 = 7-(key %7);

 for(i=0;i<TABLE\_SIZE; i++)

 {

    index=(hkey+i\*hash2)%TABLE\_SIZE;

    if(h[index]==key)

    {

      printf("value is found at index %d",index);

      break;

    }

  }

  if(i == TABLE\_SIZE)

    printf("\n value is not found\n");

}

void display()

{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE\_SIZE; i++)

    printf("\nat index %d \t value =  %d",i,h[i]);

}

int main()

{

    int opt,i;

    while(1)

    {

        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

        scanf("%d",&opt);

        switch(opt)

        {

            case 1:

                insert();

                break;

            case 2:

                display();

                break;

            case 3:

                search();

                break;

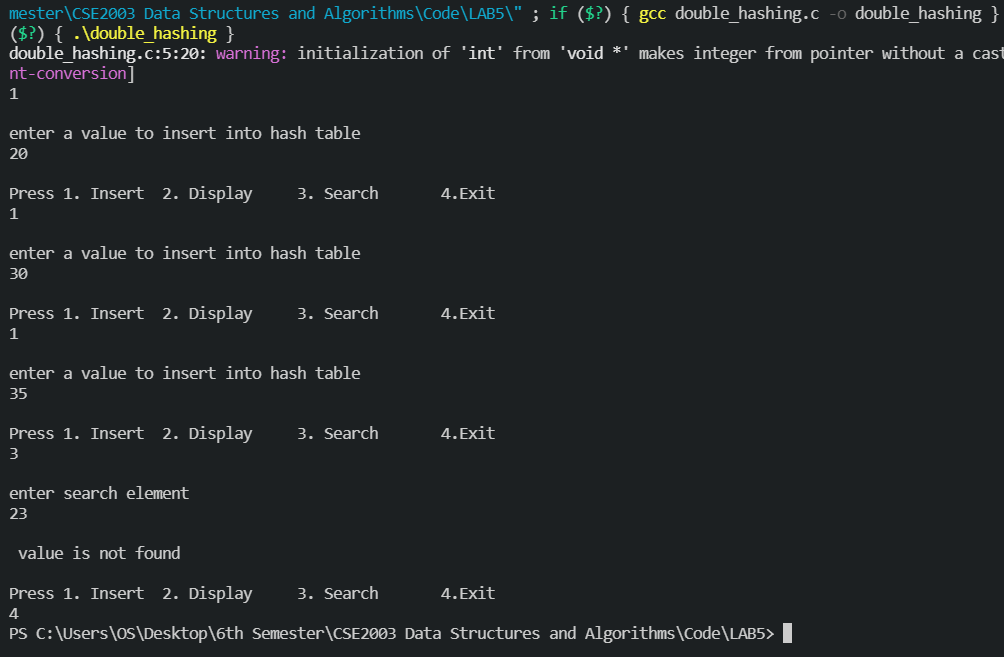
            case 4:exit(0);

        }

    }

}

**Output:**



**Conclusion**

Thus we have looked at min heaps and max heaps in C and implemented the same and verified the results. We have also implemented the four different types of hashing and verified the results for the same as well. Hence, the experiment is complete.

*Neh Samir Joshi*

*19BEC1112*