1: Write a C Program to implement linear search and binary search

Program:

#include <stdio.h>

int linearsearch(int ar[], int l, int m)

{

int i;

for (i = 0; i < l; i++)

if (ar[i] == m)

return i;

return -1;

}

int binarySearch(int ar[], int l, int r, int m)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (ar[mid] == m)

return mid;

if (ar[mid] > m)

return binarySearch(ar, l, mid - 1, m);

return binarySearch(ar, mid + 1, r, m);

}

return -1;

}

int main()

{

int l;

printf("Enter the size of the array\n");

scanf("%d",&l);

int a[l],i,m,result;

printf("Enter the elements\n");

for(i=0;i<l;i++)

scanf("%d",&a[i]);

printf("Enter the number you want to locate using linear search\n");

scanf("%d",&m);

result = linearsearch(a, l, m);

(result == -1)? printf("Element is not present in array") : printf("Element is present at array index %d", result);

printf("\nEnter the number you want to locate using binary search\n");

scanf("%d",&m);

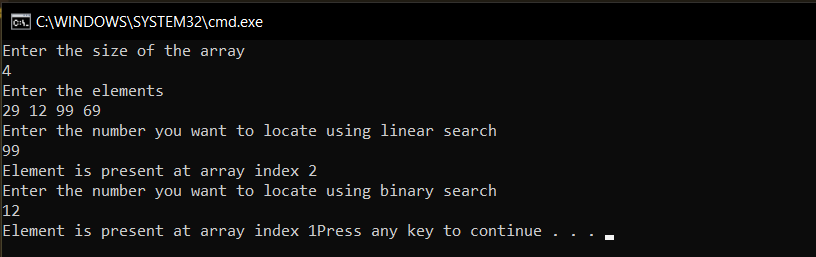
result = binarySearch(a, 0, l - 1, m);

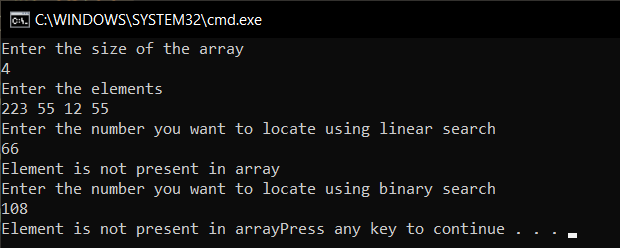
(result == -1) ? printf("Element is not present in array")

: printf("Element is present at array index %d",result);

return 0;

}





2.Write a program to implement closed hashing with linear probing

* Insert a record
* Delete a record
* Search a record
* Display table

Program:

#include <stdio.h>

#include<stdlib.h>

#define MAX 11

enum type\_of\_record {EMPTY, DELETED,OCCUPIED};

struct company

{

int id;

char name[20];

int mobile\_no;

};

struct Record

{

struct company info;

enum type\_of\_record status;

};

void insert( struct company employeee, struct Record table[]);

int search(int key, struct Record table[]);

void del(int key, struct Record table[]);

void display(struct Record table[]);

int hash(int key);

int main()

{

struct Record table[MAX];

struct company employeee;

int i,key,choice;

for( i=0; i<=MAX-1; i++ )

table[i].status = EMPTY;

while(1)

{

printf("1. Insert\n");

printf("2. Search\n");

printf("3. Delete\n");

printf("4. Display\n");

printf("5. Exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1 :

printf("Enter id, name, mobile\_no : ");

scanf("%d%s%d",&employeee.id,employeee.name,&employeee.mobile\_no);

insert(employeee, table);

break;

case 2 :

printf("Enter a key to be searched : ");

scanf("%d", &key);

i = search(key, table);

if( i==-1)

printf("Key not found\n");

else

printf("Key found at index %d\n", i);

break;

case 3:

printf("Enter a key to be deleted\n");

scanf("%d",&key);

del(key, table);

break;

case 4:

display(table);

break;

case 5:

exit(1);

}

}

}

int search(int key, struct Record table[])

{

int i, h, location;

h = hash(key);

location = h;

for( i=1; i!=MAX-1; i++ )

{

if( table[location].status == EMPTY )

return -1;

if( table[location].info.id == key)

return location;

location = ( h + i ) % MAX;

}

return -1;

}

void insert( struct company employeee, struct Record table[] )

{

int i, location, h;

int key = employeee.id; //Extract key from the record

h = hash(key);

location = h;

for( i=1; i!=MAX-1; i++ )

{

if(table[location].status == EMPTY || table[location].status == DELETED)

{

table[location].info = employeee;

table[location].status = OCCUPIED;

printf("Record inserted\n\n");

return;

}

if(table[location].info.id == key)

{

printf("Duplicate key\n\n");

return;

}

location = ( h + i) % MAX;

}

printf("Record can't be inserted : Table overFlow\n\n");

}

void display(struct Record table[])

{

int i;

for(i=0; i<MAX; i++)

{

printf("[%d] : ",i );

if(table[i].status==OCCUPIED)

{

printf("Occupied : %d %s",table[i].info.id, table[i].info.name);

printf(" %d\n", table[i].info.mobile\_no);

}

else if(table[i].status==DELETED)

printf("Deleted\n");

else

printf("Empty\n");

}

}

void del(int key, struct Record table[])

{

int location = search(key, table);

if(location == -1)

printf("Key not found\n");

else

table[location].status = DELETED;

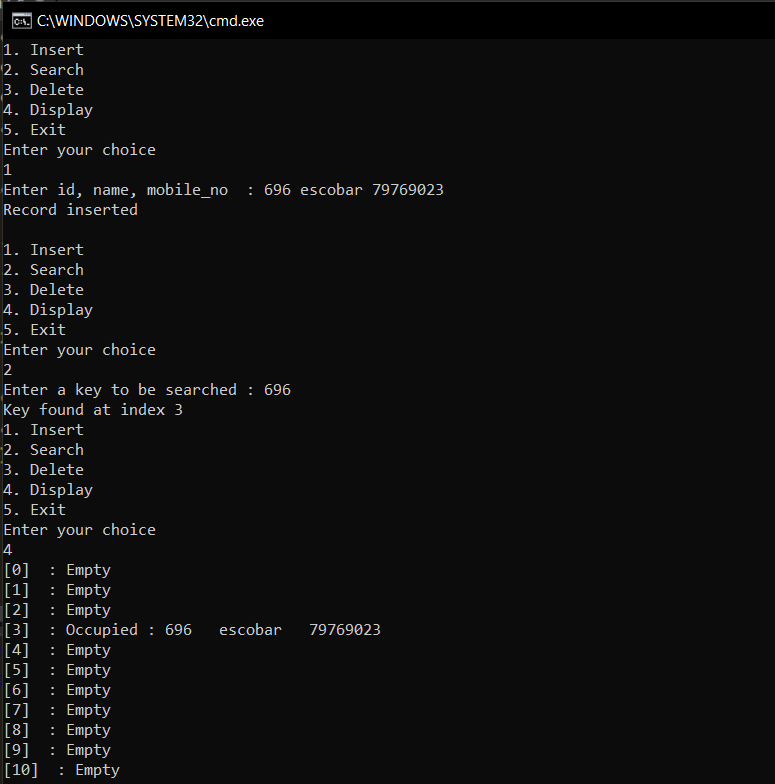
}

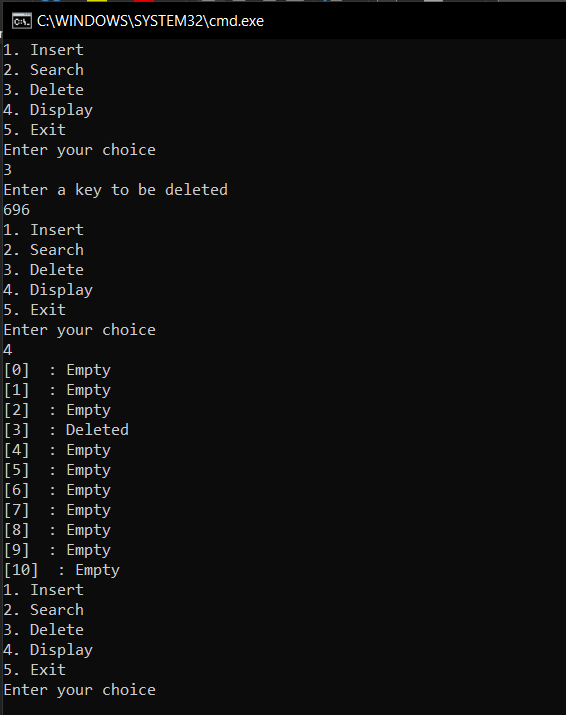
int hash(int key)

{

return (key%MAX);

}





3.Write a program to implement closed hashing with quadratic probing

* Insert a record
* Delete a record
* Search a record
* Display table

Program:

#include <stdio.h>

#include<stdlib.h>

#define MAX 11

enum type\_of\_record {EMPTY, DELETED,OCCUPIED};

struct company

{

int id;

char name[20];

int mobile\_no;

};

struct Record

{

struct company info;

enum type\_of\_record status;

};

void insert( struct company employee, struct Record table[]);

int search(int key, struct Record table[]);

void del(int key, struct Record table[]);

void display(struct Record table[]);

int hash(int key);

int main()

{

struct Record table[MAX];

struct company employee;

int z,key,choice;

for( z=0; z<=MAX-1; z++ )

table[z].status = EMPTY;

while(1)

{

printf("1. Insert\n");

printf("2. Search\n");

printf("3. Delete\n");

printf("4. Display\n");

printf("5. Exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1 :

printf("Enter id, name, mobile\_no : ");

scanf("%d%s%d",&employee.id,employee.name,&employee.mobile\_no);

insert(employee, table);

break;

case 2 :

printf("Enter a key to be searched : ");

scanf("%d", &key); z = search(key, table);

if( z==-1)

printf("Key not found\n");

else

printf("Key found at index %d\n", z);

break;

case 3:

printf("Enter a key to be deleted\n");

scanf("%d",&key);

del(key, table);

break;

case 4:

display(table);

break;

case 5:

exit(1);

}

}

}

int search(int key, struct Record table[])

{

int z, h, location;

h = hash(key);

location = h;

for( z=1; z!=MAX-1; z++ )

{

if( table[location].status == EMPTY )

return -1;

if( table[location].info.id == key)

return location;

location = ( h + (z\*z) ) % MAX;

}

return -1;

}

void insert( struct company employee, struct Record table[] )

{

int z, location, h;

int key = employee.id; //Extract key from the record

h = hash(key);

location = h;

for( z=1; z!=MAX-1; z++ )

{

if(table[location].status == EMPTY || table[location].status == DELETED)

{

table[location].info = employee;

table[location].status = OCCUPIED;

printf("Record inserted\n\n");

return;

}

if(table[location].info.id == key)

{

printf("Duplicate key\n\n");

return;

}

location = ( h + (z\*z)) % MAX;

}

printf("Record can't be inserted : Table overFlow\n\n");

}

void display(struct Record table[])

{

int z;

for(z=0; z<MAX; z++)

{

printf("[%d] : ",z );

if(table[z].status==OCCUPIED)

{

printf("Occupied : %d %s",table[z].info.id, table[z].info.name);

printf(" %d\n", table[z].info.mobile\_no);

}

else if(table[z].status==DELETED)

printf("Deleted\n");

else

printf("Empty\n");

}

}

void del(int key, struct Record table[])

{

int location = search(key, table);

if(location == -1)

printf("Key not found\n");

else

table[location].status = DELETED;

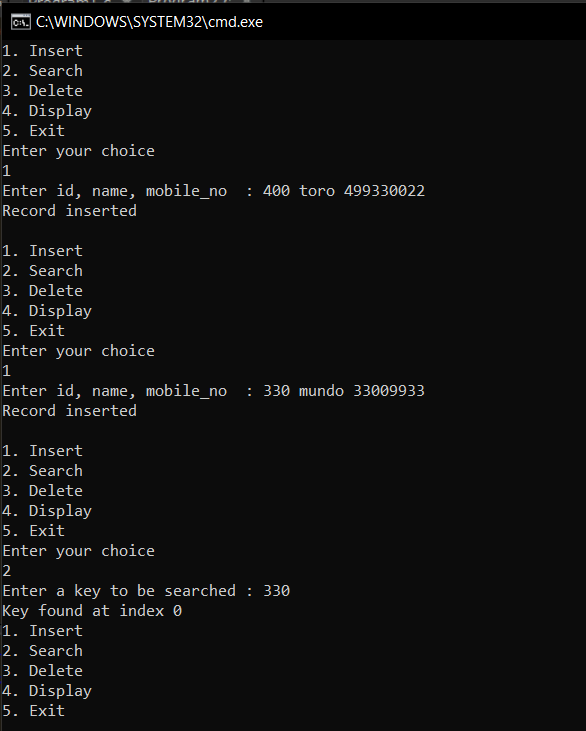
}

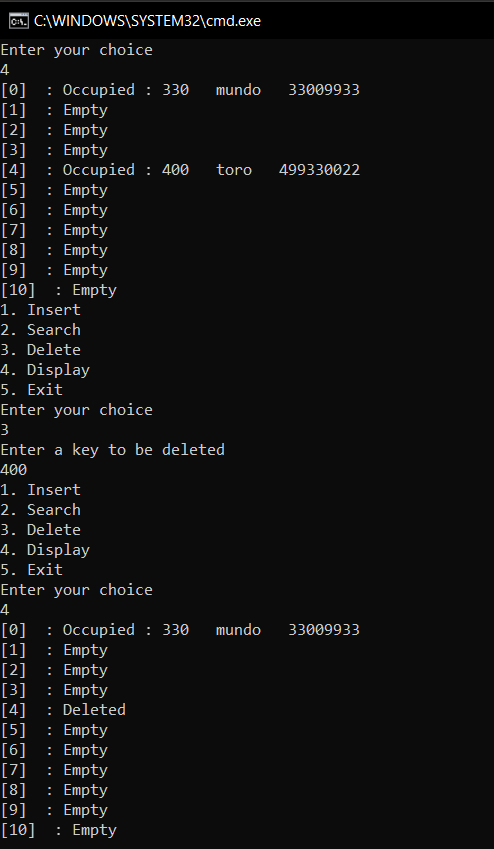
int hash(int key)

{

return (key%MAX);

}





4.Write a program to implement closed hashing with double hashing

• Insert a record

• Search a record

• Delete a record

• Display table

Program:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct data

{

int m;

int vl;

};

struct hashtable\_item

{

int fg;

struct data \*item;

};

struct hashtable\_item \*array;

int max = 7;

int size = 0;

int prime = 3;

int hashcode1(int m)

{

return (m % max);

}

int hashcode2(int m)

{

return (prime - (m % prime));

}

void insert(int m, int vl)

{

int hash1 = hashcode1(m);

int hash2 = hashcode2(m);

int index = hash1;

struct data \*new\_item = (struct data\*)malloc(sizeof(struct data));

new\_item->m = m;

new\_item->vl = vl;

if (size == max)

{

printf("\n Hash Table is full, cannot insert more items \n");

return;

}

while (array[index].fg == 1)

{

if (array[index].item->m == m)

{

printf("\n Key already present, hence updating its value \n");

array[index].item->vl = vl;

return;

}

index = (index + hash2) % max;

if (index == hash1)

{

printf("\n Add is failed \n");

return;

}

printf("\n probing \n");

}

array[index].item = new\_item;

array[index].fg = 1;

size++;

printf("\n Key (%d) has been inserted \n", m);

}

void remove\_element(int m)

{

int hash1 = hashcode1(m);

int hash2 = hashcode2(m);

int index = hash1;

if (size == 0)

{

printf("\n Hash Table is empty \n");

return;

}

while (array[index].fg != 0)

{

if (array[index].fg == 1 && array[index].item->m == m)

{

array[index].item = NULL;

array[index].fg = 2;

size--;

printf("\n Key (%d) has been removed \n", m);

return;

}

index = (index + hash2) % max;

if (index == hash1)

{

break;

}

}

printf("\n Key (%d) does not exist \n", m);

}

int size\_of\_hashtable()

{

return size;

}

void display()

{

int i;

for (i = 0; i < max; i++)

{

if (array[i].fg != 1)

{

printf("\n Array[%d] has no elements ", i);

}

else

{

printf("\n Array[%d] has elements \n Key (%d) and Value (%d) ", i, array[i].item->m, array[i].item->vl);

}

}

}

int search(int x)

{

int i,fg=0;

for (i = 0; i < max; i++)

{

if (array[i].fg != 1)

fg=0;

else

{

if(array[i].item->vl==x)

fg= 1;

}

}

return fg;

}

void init\_array()

{

int i;

for(i = 0; i < max; i++)

{

array[i].item = NULL;

array[i].fg = 0;

}

}

int main()

{

int choice, m, vl,c, x;

array = (struct hashtable\_item\*)malloc(max \* sizeof(struct hashtable\_item));

init\_array();

do

{

printf("1.Insert"

"\n2.Delete"

"\n3.Search"

"\n4.Display"

"\nPlease enter your choice:");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Inserting element in Hash Table\n");

printf("Enter Key and Value:\t");

scanf("%d %d", &m, &vl);

insert(m, vl);

break;

case 2:

printf("Deleting in Hash Table \n Enter the Key to delete-:");

scanf("%d", &m);

remove\_element(m);

break;

case 3:

printf("Enter the value of the element you want to search\n");

scanf("%d",&x);

if(search(x))

printf("Element exists\n");

else

printf("Element does not exist\n");

break;

case 4:

display();

break;

default:

printf("Wrong Input\n");

}

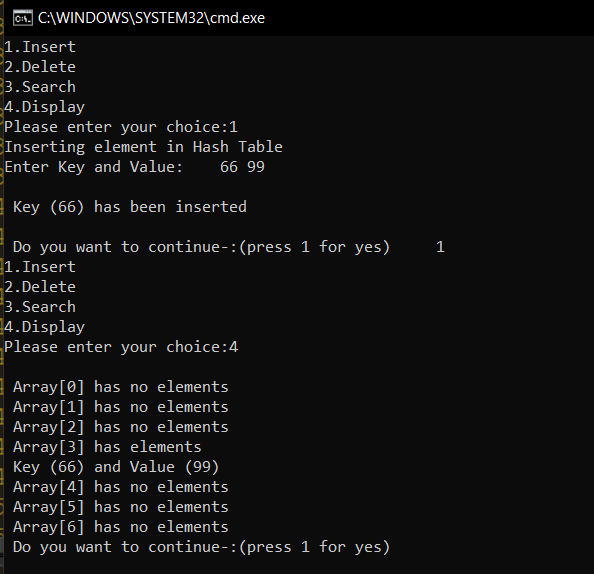
printf("\n Do you want to continue-:(press 1 for yes)\t");

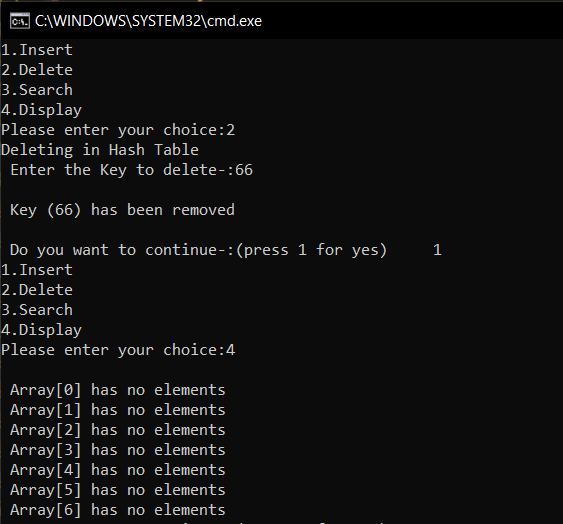
scanf("%d", &c);

}

while(c == 1);

}





5.Write a program to implement separate chaining

• Insert a record

• Search a record

• Delete a record

• Display table

Program:

#include<stdio.h>

#include<stdlib.h>

#define size 7

struct n

{

int d;

struct n \*next;

};

struct n \*chain[size];

void init()

{

int i;

for(i = 0; i < size; i++)

chain[i] = NULL;

}

void remove\_key(int key)

{

int index=key%size;

struct n \*temp=chain[index],\*point;

if(chain[index]->d==key)

chain[index]=NULL;

else

{

while(temp->next!=NULL)

{

if(temp->next->d==key)

{

point=temp->next;

temp->next=point->next;

free(point);

}

temp=temp->next;

}

}

}

void insert(int value)

{

struct n \*newNode =(struct n\*)malloc(sizeof(struct n));

newNode->d = value;

newNode->next = NULL;

int key = value % size;

if(chain[key] == NULL)

chain[key] = newNode;

else

{

struct n \*temp = chain[key];

while(temp->next)

{

temp = temp->next;

}

temp->next = newNode;

}

}

void search(int x)

{

int i,flag=0;

for(i = 0; i < size; i++)

{

struct n \*temp = chain[i];

while(temp)

{

if(temp->d==x)

{

flag=1;

break;

}

temp = temp->next;

}

if(flag)

break;

}

if(flag)

printf("Element exist\n");

else

printf("Element does not exist\n");

}

void print()

{

int i;

for(i = 0; i < size; i++)

{

struct n \*temp = chain[i];

printf("chain[%d]-->",i);

while(temp)

{

printf("%d -->",temp->d);

temp = temp->next;

}

printf("NULL\n");

}

}

int main()

{

init();

int op,x;

do

{

printf("1.Insert \n2.Delete \n3.Search \n4.Display \n0.Exit \nEnter your option: ");

scanf("%d",&op);

switch(op)

{

case 0:

break;

case 1:

printf("Enter the element you want to insert\n");

scanf("%d",&x);

insert(x);

break;

case 2:

printf("Enter the element you want to delete\n");

scanf("%d",&x);

remove\_key(x);

break;

case 3:

printf("Enter the element you want to search\n");

scanf("%d",&x);

search(x);

break;

case 4:

print();

break;

default:printf("Invalid input re enter\n");

break;

}

}

while(op!=0);

return 0;

}

