Date: 11.12.2023 Name: Niranjan.B
Ex. No: 12 Reg. No.: 3122225001082

UCS 2312 Data Structures Lab

Exercise 12: Implementation of Hash Table using Closed and Open addressing methods

Date of Assignment: 11.12.2023

The HashTableADT contains hash table and its size. Hash function to be used for the insertion of elements is ***x mod tableSize***. Use Separate chaining method to resolve the collision.

- void init(HashTableADT *H) – To initialize the size of Hash Table
- void insertElementL (HashTableADT *H, int x)– To insert the input key into the hash table
- int searchElement(HashTableADT *H, int key) – Searching an element in the hash table, if found return 1, otherwise return -1
- void displayHT(HashTableADT *H) – Display the elements in the hash table

1. Demonstrate ADT with the following test case

insert 23, 45, 69, 87, 48, 67, 54, 66, 53

2. Create another hash table ADT with following functions for open addressing methods, namely, Quadratic probing and Double Hashing.

- void insertElementL (HashTableADT *H, int x)– To insert the input key into the hash table
- void displayHT(HashTableADT *H) – Display the elements in the hash table

Note: For Double hashing, the second hash function is 7-(x%7)

Algorithm –
Algorithm: Separate Chaining (Insertion)
Input – Pointer to Hash Table, data x to be inserted
Output – void

1. Create a node.
2. node->data=x
3. h=x%size
4. ptr=H->list[h]
5. while(ptr->next!=NULL

   ptr=ptr->next

6. ptr->next=node

Algorithm: Quadratic Probing (Insertion)

Input – Pointer to Hash Table, data x to be inserted
Output – void

1. flag=1
2. for i=0 to size
   pos=(x+(i*i))%size
   if(H->table[pos]==-1
   H->table[pos]=x
   Flag=0
   Break
3. if flag==1
   print Table is Full

Algorithm: Double Hashing (Insertion)

Input – Pointer to Hash Table, data x to be inserted
Output – void

1. flag=1
2. prime=first prime number smaller than size
3. for i=0 to size
   hash2=prime-(x%prime)
   pos=((x%size)+(i*hash2))%size)
   if(H->table[pos]==-1
   H->table[pos]=x
   Flag=0
   Break
4. if flag==1
   print Table is Full

hash1.h code:

```
//separate chaining

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

struct hashtable
{
  int s;
  struct node* list[100];
};

void create (struct hashtable *H, int size)
{
  H->s = size;
  for (int i=0;i< H->s; i++)
  {
    H->list[i] = (struct node*)malloc(sizeof(struct node));
    H->list[i]->next = NULL;
  }
}
```

```
void insert (struct hashtable *H, int x)
{
  struct node* temp = (struct node*)malloc(sizeof(struct node));
  temp->next = NULL;
  temp->data = x;
  int h = x % H->s;
  struct node*  ptr = H->list[h];
  while (ptr->next!=NULL)
  {
    ptr = ptr->next;
  }
  ptr->next = temp;
}

void print (struct hashtable *H)
{
  for (int i=0;i<H->s;i++)
  {
    printf ("\n%d - ", i);
    struct node* header = H->list[i];
    struct node* ptr = header->next;
    while (ptr!=NULL)
      {
        if (ptr==header->next)
          printf (" %d ", ptr->data);
        else
          printf (" -> %d", ptr->data);
        ptr = ptr->next;
      }
  }
  printf ("\n");
}

void search (struct hashtable *H, int x)
{
  int h = x % H->s;
  struct node* ptr = H->list[h];
  while (ptr->next!=NULL)
    {
      ptr = ptr->next;
      if (ptr->data==x)
      {
        printf ("\nElement %d found.\n", x);
        return;
      }
    }
  printf ("\nElement %d not found.\n", x);
}
```

hash1.c code:

```
//separate chaining

#include <stdio.h>
```

```
#include <stdlib.h>
#include "hash1.h"

void main ()
{
  struct hashtable* H = (struct hashtable *)malloc(sizeof(struct
hashtable));
  create (H,10);
  insert(H,23);
  insert(H,45);
  insert(H,69);
  insert(H,87);
  insert(H,48);
  insert(H,67);
  insert(H,54);
  insert(H,66);
  insert(H,53);
  print (H);
  search (H,45);
  search (H,67);
  search (H,12);
}
```

hash2i.h code:

```
struct hashtable
{
    int size;
    int table[100];
};

void create (struct hashtable *H, int size)
{
  H->size = size;
  for (int i=0;i< H->size; i++)
  {
    H->table[i]=-1;
  }
}

void insert (struct hashtable *H, int x)
{
  int pos,flag=1;
  for(int i=0;i<H->size;i++)
  {
    pos=(x+(i*i))%(H->size);
    if(H->table[pos]==-1)
    {
        H->table[pos]=x;
        flag=0;
        break;
    }
  }
  if(flag)
  {
```

```
    printf("Hash Table is full.\n");
  }
}


void display (struct hashtable *H)
{
  printf("Hash Table Elements : ");
  for(int i=0;i<H->size;i++)
  {
    if(H->table[i]!=-1)
    {
        printf("%d ",H->table[i]);
    }
  }
  printf("\n");
}
```

hash2i.c code:

```
//double hashing

#include <stdio.h>
#include <stdlib.h>
#include "hash2i.h"

void main ()
{
  struct hashtable* H = (struct hashtable *)malloc(sizeof(struct
hashtable));
  int choice=1,size,data;
  printf("Size = ");
  scanf("%d",&size);
  create(H,size);
  while(choice)
  {
    printf("\n1.Insert\n2.Print\nChoice : ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
        printf("Element = ");
        scanf("%d",&data);
        insert(H,data);
        break;
        case 2:
        display(H);
        break;
    }
  }
}
```

hash2ii.h code:

```
#include <stdio.h>

struct hashtable
{
    int size;
    int table[100];
};

void create (struct hashtable *H, int size)
{
  H->size = size;
  for (int i=0;i< H->size; i++)
  {
    H->table[i]=-1;
  }
}

int primeNo (struct hashtable *H)
{
    int c=0;
    for(int i=H->size-1;i=0;i--)
    {
        c=0;
        for(int j=1;j<=i;j++)
        {
            if(i%j==0)
            {
                ++c;
            }
        }
        if(c==2)
        {
            return i;
        }
    }
}

void insert (struct hashtable *H, int x)
{
  int pos,flag=1,hash2,prime;
  prime=primeNo(H);
  for(int i=0;i<H->size;i++)
  {
    hash2=prime-(x%prime);
    pos=((x%H->size)+(i*hash2))%(H->size);
    if(H->table[pos]==-1)
    {
        H->table[pos]=x;
        flag=0;
        break;
    }
  }
  if(flag)
```

```
  {
    printf("Hash Table is full.\n");
  }
}


void display (struct hashtable *H)
{
  printf("Hash Table Elements : ");
  for(int i=0;i<H->size;i++)
  {
    if(H->table[i]!=-1)
    {
        printf("%d ",H->table[i]);
    }
  }
  printf("\n");
}
```

hash2ii.c code:

```
//double hashing

#include <stdio.h>
#include <stdlib.h>
#include "hash2i.h"

void main ()
{
  struct hashtable* H = (struct hashtable *)malloc(sizeof(struct
hashtable));
  int choice=1,size,data;
  printf("Size = ");
  scanf("%d",&size);
  create(H,size);
  while(choice)
  {
    printf("\n1.Insert\n2.Print\nChoice : ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
        printf("Element = ");
        scanf("%d",&data);
        insert(H,data);
        break;
        case 2:
        display(H);
        break;
    }
  }
}
```

Output Screen:

Separate Chaining-

```
PS D:\College\Sem 3\Data Structures\Hash Table> gcc hash1.c
PS D:\College\Sem 3\Data Structures\Hash Table> ./a.exe

0 -
1 -
2 -
3 -  23  -> 53
4 -  54
5 -  45
6 -  66
7 -  87  -> 67
8 -  48
9 -  69

Element 45 found.

Element 67 found.

Element 12 not found.
PS D:\College\Sem 3\Data Structures\Hash Table>
```

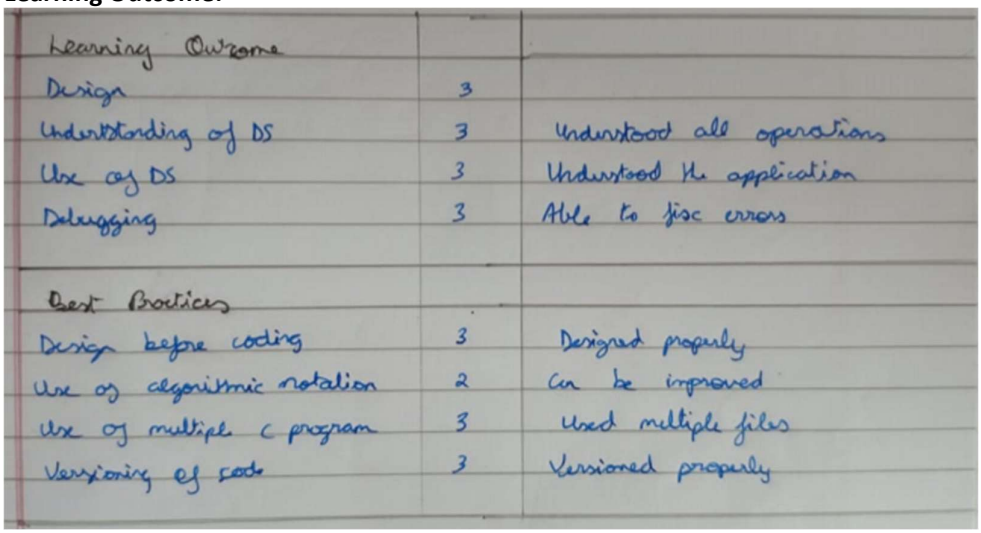
Quadratic Probing-

```
PS D:\College\Sem 3\Data Structures\Hash Table> gcc hash2i.c
PS D:\College\Sem 3\Data Structures\Hash Table> ./a.exe
Size = 10

1.Insert
2.Print
Choice : 1
Element = 23

1.Insert
2.Print
Choice : 1
Element = 45

1.Insert
2.Print
Choice : 1
Element = 69

1.Insert
2.Print
Choice : 1
Element = 87

1.Insert
2.Print
Choice : 1
Element = 48

1.Insert
2.Print
Choice : 1
Element = 67

1.Insert
2.Print
Choice : 1
Element = 54

1.Insert
2.Print
Choice : 1
Element = 66

1.Insert
2.Print
Choice : 1
Element = 53

1.Insert
2.Print
Choice : 2
Hash Table Elements : 67 53 23 54 45 66 87 48 69
```

Double Hashing-

```
PS D:\College\Sem 3\Data Structures\Hash Table> gcc hash2ii.c
PS D:\College\Sem 3\Data Structures\Hash Table> ./a.exe
Size = 10

1.Insert
2.Print
Choice : 1
Element = 23

1.Insert
2.Print
Choice : 1
Element = 45

1.Insert
2.Print
Choice : 1
Element = 69

1.Insert
2.Print
Choice : 1
Element = 87

1.Insert
2.Print
Choice : 1
Element = 48

1.Insert
2.Print
Choice : 1
Element = 67

1.Insert
2.Print
Choice : 1
Element = 54

1.Insert
2.Print
Choice : 1
Element = 66

1.Insert
2.Print
Choice : 1
Element = 53

1.Insert
2.Print
Choice : 2
Hash Table Elements : 67 53 23 54 45 66 87 48 69
```

Learning Outcome:



Learning Outcome		
Design	3	
Understanding of DS	3	Understood all operations
Use of DS	3	Understood the application
Debugging	3	Able to fix errors
Best Practices		
Design before coding	3	Designed properly
Use of algorithmic notation	2	Can be improved
Use of multiple c program	3	Used multiple files
Versioning of code	3	Versioned properly