1. **Activity: Implement a hash function on student SAP-ID and categorised them in to 10 families based on last 3 digits**

**Algorithm:**

1. Get Sap ID from user
2. Pass Sap-Id to the hash function
   1. N = key % 1000
   2. While n > 0
      1. Dig = n%10
      2. Sum += dig
      3. N = n/10
   3. If sum > 9, n = sum and goto step b
   4. Return sum
3. Print the returned index value

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Index | Pass/Fail |
| 5000423 | 9 | Pass |
| 5000425 | 2 | Pass |
| 5000111 | 3 | Pass |
| 5000182 | 2 | Pass |

**Program:**

#include <stdio.h>

#include <ctype.h>

//Hash Function

int hash(int key){

    int h, sum;

    h = key % 1000;

    LP: sum = 0;

//Loop to find sum of all digits of n

    while (h > 0)

    {

        int dig = h % 10;

        sum += dig;

        h = h/10;

    }

//Check if the sum is reiterable

    if (sum > 9)

    {

        h = sum;

        goto LP;

    }

    return sum;

}

int main(){

    int h, k;

//Insert 2 sample data

    h = 5000423;

    k = hash(h);

    printf("Key value: %d; Index: %d\n", h, k);

    h = 5000425;

    k = hash(h);

    printf("Key value: %d; Index: %d\n", h, k);

//Loop for user input

    int cond = 1;

    while(cond){

        printf("Do you want to enter SAP-ID: ");

        char ch;

        scanf("%c", &ch);

        switch (tolower(ch))

        {

        case 'y':

            printf("Enter SAP-ID: ");

            int sap;

            scanf("%d", &sap);

            printf("Key value: %d; Index: %d\n", sap, hash(sap));

            break;

        case 'n':

            cond = 0;

            break;

        default:

            break;

        }

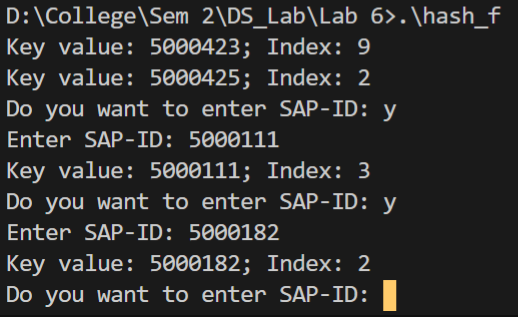
        while((getchar()) != '\n');

    }

    return 0;

}

**Screenshot of compilation and execution:**

****

1. **Activity: Implement a hash table using arrays and linear probing as collision avoidance strategy.**

**Algorithm:**

1. Hash function same as defines in Experiment 1
2. Insert function:
   1. Initial\_index = hash(key)
   2. Index = initial\_index
   3. While(True)
      1. If index == tab\_size, index = 0
      2. If arr[index] is empty, insert element at that index
      3. Else index++
         1. If index == init\_index, return
3. Search function:
   1. Initial\_index = hash(key)
   2. Index = initial\_index
   3. While(True)
      1. If index == tab\_size, index = 0
      2. If arr[index] is the desired element, return index
      3. Else index++
         1. If index == init\_index, return
4. Delete function:
   1. Index = Search(key)
   2. if arr[in] is a valid array variable, set arr[in] as empty

**Program:**

#include <stdio.h>

#include <ctype.h>

//Hash Function

int hash(int key){

    int h, sum;

    h = key % 1000;

    LP: sum = 0;

//Loop to find sum of all digits of n

    while (h > 0)

    {

        int dig = h % 10;

        sum += dig;

        h = h/10;

    }

//Check if the sum is reiterable

    if (sum > 9)

    {

        h = sum;

        goto LP;

    }

    return sum;

}

int main(){

    int h, k;

//Insert 2 sample data

    h = 5000423;

    k = hash(h);

    printf("Key value: %d; Index: %d\n", h, k);

    h = 5000425;

    k = hash(h);

    printf("Key value: %d; Index: %d\n", h, k);

//Loop for user input

    int cond = 1;

    while(cond){

        printf("Do you want to enter SAP-ID: ");

        char ch;

        scanf("%c", &ch);

        switch (tolower(ch))

        {

        case 'y':

            printf("Enter SAP-ID: ");

            int sap;

            scanf("%d", &sap);

            printf("Key value: %d; Index: %d\n", sap, hash(sap));

            break;

        case 'n':

            cond = 0;

            break;

        default:

            break;

        }

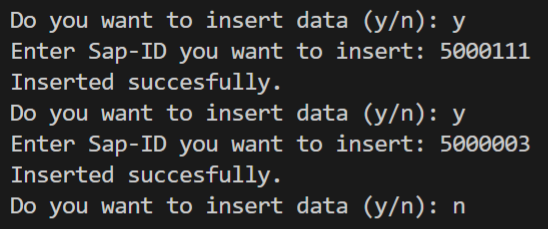
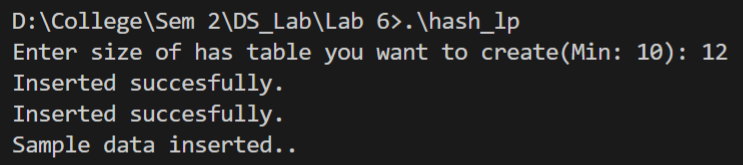
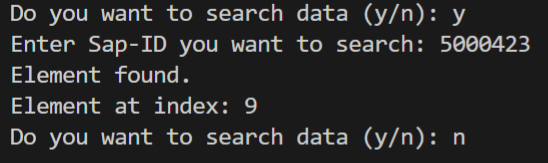
        while((getchar()) != '\n');

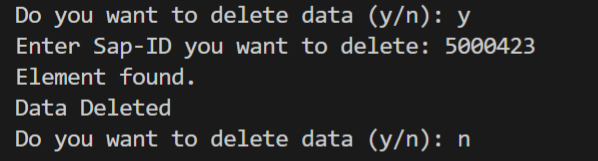
    }

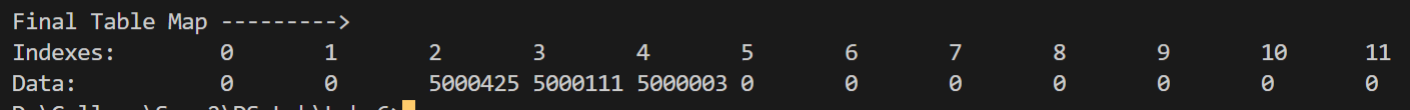
    return 0;

}

**Screenshot of compilation and execution:**

** **

****

****

1. **Activity: Implement a hash table using arrays and separate chaining as collision avoidance strategy.**

**Algorithm:**

1. Hash function same as defines in Experiment 1
2. Insert function:
   1. index = hash(key)
   2. Temp->data = key, temp->next = NULL
   3. If table[index] == NULL, table[index] = temp
   4. Else
      1. For(ptr = table[index]; ptr->next != NULL; ptr = ptr->next)
      2. Ptr->next = temp
3. Search function:
   1. index = hash(key)
   2. if table[index] == NULL, print data not found
   3. count = 1
   4. For (ptr = table[index]; ptr->data != key; ptr = ptr->next, count++)
      1. If ptr->next == NULL, print Data not found
   5. Print element found at index (count) of list in index (index)
4. Delete function:
   1. Index = hash(key)
   2. If table[index] == NULL, print Data not Found, return
   3. If table[index]->data == Key
      1. If table[index]->next == NULL, table[index] = NULL, return
      2. Else,
         1. Temp = table[index]
         2. Table[index] = temp->next
         3. Free(temp)
         4. return
   4. If table[index]->next == NULL print Data not found, return
   5. for (ptr = table[index]; ptr->next->data != key; ptr = ptr->next)
      1. if (ptr->next->next == NULL) print Data not found, return
   6. temp = ptr->next
   7. ptr->next = temp->next
   8. free(temp)

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

struct Node{

    int data;

    struct Node \*next;

} \*ptr = NULL, \*table[10] = {NULL};

//Hash function

int hash(int key){

    int h, sum;

    h = key % 1000;

    LP: sum = 0;

    while (h > 0)

    {

        int dig = h % 10;

        sum += dig;

        h = h/10;

    }

    if (sum > 9)

    {

        h = sum;

        goto LP;

    }

    return sum;

}

void insert(int key){

    int index = hash(key);

    struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));

    temp->data  = key;

    temp->next = NULL;

    if(table[index] == NULL){

        table[index] = temp;

    }

    else{

        for(ptr = table[index]; ptr->next != NULL; ptr = ptr->next);

        ptr->next = temp;

    }

}

void search(int key){

    int index = hash(key);

    if (table[index] == NULL)

    {

        printf("Data Not Found\n");

        return;

    }

    int count = 1;

    for (ptr = table[index]; ptr->data != key; ptr = ptr->next, count ++){

        if (ptr->next == NULL)

        {

            printf("Data Not Found\n");

            return;

        }

    }

    printf("Element found at position %d of list in index %d\n", count, index);

}

void delete(int key){

    int index = hash(key);

    struct Node \*temp;

    if (table[index] == NULL)

    {

        printf("Data Not Found\n");

        return;

    }

    if(table[index]->data == key){

        if (table[index]->next == NULL)

        {

            table[index] = NULL;

            printf("Deleted\n");

            return;

        }

        temp = table[index];

        table[index] = temp->next;

        free(temp);

        printf("Deleted\n");

        return;

    }

    if (table[index]->next == NULL)

    {

        printf("Data Not found\n");

        return;

    }

    for (ptr = table[index]; ptr->next->data != key; ptr = ptr->next){

        if (ptr->next->next == NULL)

        {

            printf("Data Not Found\n");

            return;

        }

    }

    temp = ptr->next;

    ptr->next = temp->next;

    free(temp);

    printf("Deleted\n");

}

void print\_table(){

    printf("Index\tData\n");

    for (int i = 0; i < 10; i++)

    {

        printf("%d\t", i);

        for (ptr = table[i]; ptr != NULL; ptr = ptr->next)

        {

            printf("%d\t", ptr->data);

        }

        printf("\n");

    }

}

int main(){

    insert(5000423);

    insert(5000432);

    insert(5000425);

    char choice;

    printf("Sample data inserted..\n\n");

    while(1){

        printf("Do you want to insert data (y/n): ");

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y':

            printf("Enter Sap-ID you want to insert: ");

            int sap;

            scanf("%d", &sap);

            insert(sap);

            break;

        case 'n':

            goto Exit\_1;

            break;

        default:

            printf("Invalid selection.\n");

            break;

        }

        while((getchar()) != '\n');

    }

    Exit\_1:printf("\n");

    while(1){

        printf("Do you want to search data (y/n): ");

        while((getchar()) != '\n');

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y':

            printf("Enter Sap-ID you want to search: ");

            int sap;

            scanf("%d", &sap);

            search(sap);

            break;

        case 'n':

            goto Exit\_2;

            break;

        default:

            printf("Invalid selection.\n");

            break;

        }

    }

    Exit\_2:printf("\n");

    while(1){

        printf("Do you want to delete data (y/n): ");

        while((getchar()) != '\n');

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y':

            printf("Enter Sap-ID you want to delete: ");

            int sap;

            scanf("%d", &sap);

            delete(sap);

            break;

        case 'n':

            goto Exit\_3;

            break;

        default:

            printf("Invalid selection.\n");

            break;

        }

    }

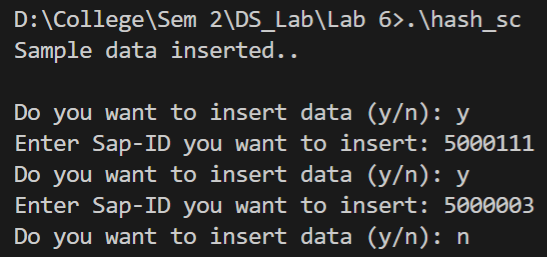
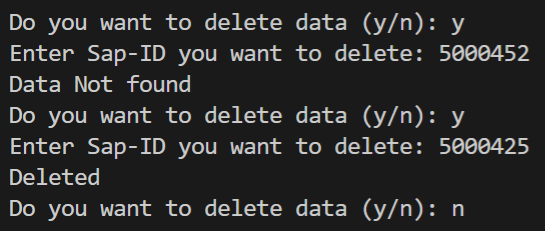
    Exit\_3:printf("\n");

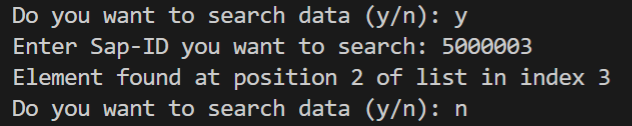
    print\_table();

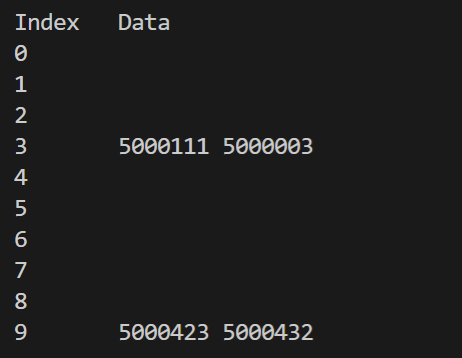
    return 0;

}

**Screenshot of compilation and execution:**

** **

****

****