Hashing Type Linked list

จัดทำโดย

1.	นางสาวคงสิริ ปานชื่น	รหัสประจำตัวนิสิต	6530200045
2.	นางสาวทิพวรรณ งบกระโทก	รหัสประจำตัวนิสิต	6530200193
3.	นายธนพล ยุราวรรณ	รหัสประจำตัวนิสิต	6530200215
4.	นายวัชระพล นาเมือง	รหัสประจำตัวนิสิต	6530200240
5.	นายปรวภัทร มุธะรพัฒน์	รหัสประจำตัวนิสิต	6530200266
6.	นายพีระเมศ จุกกษัตริย์	รหัสประจำตัวนิสิต	6530200304
7.	นางสาวปรียดา เกษตรภิบาล	รหัสประจำตัวนิสิต	6530200690

เสนอ

อาจารย์จิรวรรณ เจริญสุข

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา

วิชา Data Structure and Algorithms (โครงสร้างข้อมูล)

รหัสวิชา 01418231 หมู่เรียนบรรยาย 800

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตศรีราชา

Code

สามารถดู code เพิ่มเติมได้จาก https://onlinegdb.com/SuV8m1bBn

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>



int size;
int separateChain(){

    bool isPositiveInteger(int number) {
    return number > 0;
    }

    printf("\n\t\tSeparate Chain");
    printf("\n------------------------------------------------");//เช็คerror
    do {
        printf("\nEnter table size (must be greater than 0): ");
        if (scanf("%d", &size) != 1 || !isPositiveInteger(size)) {
            printf("Invalid table size. Please enter a positive integer.\n");
            while (getchar() != '\n');
        }
```

```
    } while (!isPositiveInteger(size));



    // ตรวจสอบค่าที่รับเข้ามา
    while (size <= 0) {
        printf("Invalid size!\n");
        printf("Please enter again in positive integer : "); // รับค่าอีกครั้งที่เป็นตัวเลขจำนวนเต็มบวก
        scanf("%d", &size);
    }

    return size;
}



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



// โครงสร้างสำหรับตารางแฮช
struct hashTable {
```

```
    struct node* head;
};



// ฟังก์ชันคำนวณค่าแฮช
int hashCode(int key) {
    return key % size;
}



// ฟังก์ชันเพิ่มค่าในตารางแฮช
void insert(struct hashTable* hash, int key) {
    int index = hashCode(key);
    struct node* newNode = (struct node*)malloc(sizeof(struct node));//จองพื้นที่
    newNode->data = key;
    newNode->next = NULL;

    // ถ้าไม่มีโหนดในช่องแฮชนี้
    if (hash[index].head == NULL) {
        hash[index].head = newNode;
    }
    // มีโหนดอยู่แล้ว ให้เพิ่มโหนดใหม่ลงไปที่สุดของลิงก์ลิสต์
    else {
```

```
        struct node* current = hash[index].head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}



int search(struct hashTable* hash,int key){
    int index = hashCode(key);
    struct node* current = hash[index].head;



    while (current != NULL) {
        if (current->data == key) {
            return index; // เจอเลขที่ต้องการ
        }
        current = current->next;
    }
    //printf("test");
```

```
    return -1; // ไม่เจอเลขที่ต้องการ
}




int delete(struct hashTable *hash,int key){
    int index = hashCode(key);
    struct node* current = hash[index].head;
    struct node *prev = NULL;

    while(current != NULL){
        if(current->data == key){
            if(prev == NULL){
            hash[index].head = current->next;
            }
            else{
                prev->next = current->next;
            }
            free(current);//คืนค่าตำแหน่งหลังจากทำงานเสร็จ
            return 1; //เจอเลขที่ต้องการ
        }
        prev = current;
        current = current->next;
```

```
    }
        printf("Can't delete - Not in the table\n");
        return 0 ; //ไม่เจอเลขที่ต้องการ
}
// ฟังก์ชันแสดงค่าในตารางแฮช
void display(struct hashTable *hash) {
    printf("\n\t\t Hash Table ");
    printf("\n-------------------------------------------------\n");
    for (int i = 0; i < size; i++) {
        printf("Bucket[%d] -> ", i);
        struct node *current = hash[i].head;
        while (current != NULL) {
            printf("%d -> ", current->data);
            current = current->next;
        }
        printf("NULL\n");
    }
}

// โครงสร้างข้อมูลสำหรับเก็บค่าในแต่ละช่องของตาราง
struct NodeL {
```

```
    int key;            // ค่าที่จะเก็บในลิงค์ลิสต์
    struct NodeL* next;  // พอยน์เตอร์ไปยังโหนดถัดไปในลิงค์ลิสต์
};

// ฟังก์ชันสำหรับ Linear Probing
int linearProbing(int key, struct NodeL* arr[], int size) {
    int index = key % size;
    struct NodeL* newNode = (struct NodeL*)malloc(sizeof(struct NodeL));
    newNode->key = key;
    newNode->next = NULL;



    if (arr[index] == NULL) {
        arr[index] = newNode;
        return index;
    } else {
       while (arr[index] != NULL && arr[index]->key != key && arr[index]->key != -1) {
            index = (index + 1) % size ;
       }
       if (arr[index] == NULL || arr[index]->key == -1)
            size++;
            arr[index] = newNode;
        }
```

```
    }

// ฟังก์ชันสำหรับการค้นหาค่าในตาราง
int searchL(int key, struct NodeL* arr[], int size) {
    int index = key % size;
    struct NodeL* current = arr[index];

    while (current != NULL) {
        if (current->key == key) {
            return index; // คืนค่า index ที่พบ key
        }
        current = current->next;
    }

    return -1; // ไม่พบ key ในตาราง
}

// ฟังก์ชันสำหรับลบข้อมูลออกจากตาราง
void deleteL(int key, struct NodeL* arr[], int size) {
    int index = key % size;
    struct NodeL* current = arr[index];
    struct NodeL* prev = NULL;
```

```
    while (current != NULL) {
        if (current->key == key) {
            if (prev == NULL) {
                // ข้อมูลที่ต้องการลบอยู่ในหัวของลิงค์ลิสต์
                arr[index] = current->next;
            } else {
                // ข้อมูลที่ต้องการลบไม่ได้อยู่ในหัวของลิงค์ลิสต์
                prev->next = current->next;
            }
            free(current); // ลบโหนดที่มีข้อมูลที่ต้องการลบ
            return;
        }
        prev = current;
        current = current->next;
    }
    // ไม่พบข้อมูลที่ต้องการลบในตาราง
    printf("Can't delete - Number not found\n");
}



int size;


//ใช้รับค่าขนาดของ Table
```

```
int tablesize() {
    printf("\n\t\tQuadratic hashing\n");
    printf("-----------------------------------------------\n");
    printf("Please enter size of table : ");
    scanf("%d", &size);
    //printf("-----------------------------------------------\n");

    while (size <= 0) {
        printf("Invalid size\n");
        printf("Please enter again in a positive integer : ");
        scanf("%d", &size);
        //printf("-----------------------------------------------\n");
    }

    return size;
}

struct Node {
    int data;
    struct Node* next;
};
```

```
struct HashTable {
    struct Node* head;
};

int hash(int key) {
    return key % size;
}

void insertQ(struct HashTable* hashTable, int key) {

    while(1){
        printf("\n\t\tInsert");
        printf("\n--------------------------------------------------\n");
        printf("Enter a value to insert into the hash table : ");
        if (scanf("%d", &key) != 1|| key <= 0) {
            printf("\nInvalid size Please input Positive Integer\n");
            printf("--------------------------------------------------\n");
            while (getchar() != '\n');
        } else {
            while (getchar() != '\n');
            break;
        }
    }
```

```
    int index = hash(key);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = key;
    newNode->next = NULL;

    if (hashTable[index].head == NULL) {
        hashTable[index].head = newNode;
    }
    else {
        int i = 1;
        while(1) {
            int newIndex = (index + i * i) % size; //วิธีหาตำแหน่งใส่แบบ Quadratic
            if (hashTable[newIndex].head == NULL) {
                hashTable[newIndex].head = newNode;
                break;
            }
            i++;
        }
    }
}

void searchQ(struct HashTable* hashTable, int key) {
```

```
while(1){
    printf("\t\t\nSearch");
    printf("\n-----------------------------------------------\n");
    printf("Enter search element : ");
    if (scanf("%d", &key) != 1|| key <= 0) {
        printf("\nInvalid size Please input Number\n");
        printf("-----------------------------------------------\n");
        while (getchar() != '\n');
    } else {
        while (getchar() != '\n');
        break;
    }
}

int index = hash(key);
struct Node* current = hashTable[index].head;
int i = 1;
printf("-----------------------------------------------\n");
while (current != NULL) {
    if (current-> data == key) {
        printf("Key %d found at index %d\n", key, index);
        return;
```

```
        }
        int newIndex = (index + i * i) % size;
        current = hashTable[newIndex].head;
        i++;
    }
    printf("Key %d not found in the hash table\n", key);
    printf("-------------------------------------------------\n");
}

void deleteQ(struct HashTable* hashTable, int key) {

    while(1){
        printf("\t\tDelete");
        printf("\n-------------------------------------------------\n");
        printf("\nEnter the key to delete from the hash table : ");
        if (scanf("%d", &key) != 1|| key <= 0) {
            printf("\nInvalid size Please input Number\n");
            printf("-------------------------------------------------\n");
            while (getchar() != '\n');
        } else {
            while (getchar() != '\n');
            break;
        }
```

```
}


int index = hash(key);
struct Node* current = hashTable[index].head;
struct Node* prev = NULL;
printf("-------------------------------------------------\n");


while (current != NULL) {
    if (current-> data == key) {
        if (prev == NULL) {
            hashTable[index].head = current->next;
        }
        else {
            prev->next = current->next;
        }
        free(current);
        printf("Key %d deleted from the hash table\n", key);
        return;
    }
    int i = 0 ;
    int newIndex = (index + i * i) % size;
    prev = current;
    current = current->next;
```

```
        i++;
    }

    printf("Key %d not found in the hash table\n", key);
    printf("-------------------------------------------------\n");
}

void displayQ(struct HashTable* hashTable) {
    printf("-------------------------------------------------\n");
    for (int i = 0; i < size; i++) {
        printf("Index %d: ", i);
        struct Node* current = hashTable[i].head;
        while (current != NULL) {
            printf("%d -> ", current->data);
            current = current->next;
        }
        printf("NULL\n");
    }
}

//ฟังก์ชันเรียกใช้choiceที่ต้องการในแบบquadratic aka.mainแบบตอนแรก
int Quadratic() {
    tablesize();
```

```
    int c = 1;
    int key;
    int sel;
    struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct HashTable) + size *
sizeof(struct Node*));

        for (int i = 0; i < size; i++) {
       hashTable[i].head = NULL;
    }

    while(c!=0){
        //printf("\n----------------------------------------------------\n");
        printf("(Quantity of table hashing is %d)\n",size);
        printf("----------------------------------------------------\n");
        printf("\nPress 1. Insert\n       2. Search\n       3. Delete\n       4. Display\n       5. Exit\n");
        printf("\nEnter number : ");
        if (scanf("%d", &sel) != 1 || sel <= 0) {
            printf("\n----------------------------------------------------\n");
            printf("\nInvalid choice. Please enter a number between 1 and 5.\n");
            while (getchar() != '\n');
        } else {
            while (getchar() != '\n');
            c=1;
        }
```

```
switch (sel) {
case 1:
    insertQ(hashTable,0);
    displayQ(hashTable);
break;

case 2:
    searchQ(hashTable, 0);
break;

case 3:
    deleteQ(hashTable, 0);
    displayQ(hashTable);
break;

case 4:
    displayQ(hashTable);
break;

case 5:
    main();
    c=0;
}
```

```
    }


    return 0;
}




int main() {
    int choice;//รับรูปแบบการทำงาน
    int enter; //รับค่าเมนู
    int numValues; //รับค่าจำนวนข้อมูลที่จะเพิ่มในตาราง
    int searchKey; //รับค่าข้อมูลที่ต้องการค้นหา
    int removeKey;//รับค่าข้อมูลที่ต้องการลบออก


    printf("\n\t\tHashing Type");
    printf("\n--------------------------------------------------\n");
    printf("1.Separate Chain\n2.Linear Probing\n3.Quadratic Probing\n4.Exit program\n");
    printf("--------------------------------------------------");
    printf("\nEnter number : ");
    scanf("%d",&choice);


    if(choice == 1){
        //printf("\n---------- Separate Chain ----------\n");
```

```
    int size = separateChain();
    printf("-----------------------------------------------\n");
    printf("\t\ttable size is %d\n", size); // แสดงขนาดของอาร์เรย์
    struct hashTable hash[size]; //สร้างตารางแฮช
    printf("-----------------------------------------------\n");

// กำหนดทุกช่องในตารางแฮชให้เป็น NULL
    for (int i = 0; i < size; i++) {
        hash[i].head = NULL;
    }

    do{
        printf("\nPress 1. Insert\n      2. Search\n      3. Delete\n      4. Exit\n      ");
        printf("\nEnter your menu : ");

        int isValidChoice = 0;
        while (!isValidChoice) {
            if (scanf("%d", &enter) != 1 || enter < 1 || enter > 4) {
                printf("Invalid choice. (Please enter a number between 1 and 4) : ");
                while (getchar() != '\n');
            } else {
                isValidChoice = 1;
            }
```

```
}


switch(enter) {
    case 1 : //เพิ่มข้อมูลในตาราง
        printf("\n\t\tInsert");
        printf("\n-----------------------------------------------\n");
        printf("Enter number of values to insert : ");
        scanf("%d", &numValues);



        for (int i = 0; i < numValues; i++) {
            int value;
            while (1) {
                printf("Enter value %d: ", i + 1);
                if (scanf("%d", &value) != 1) {
                    printf("Invalid input. Please enter an integer.\n");
                    while (getchar() != '\n');
                }else if (value <= 0) {
                    printf("Invalid input. Please enter a positive integer.\n");
                } else {
                    insert(hash, value);
                    break;
                }
```

```
        }
    }
    display(hash);
break ;

case 2 : //ค้นหาข้อมูลจากตาราง
    printf("\n\t\t Search");
    printf("\n-------------------------------------------------\n");
    printf("Enter the number to search : ");
    scanf("%d", &searchKey);
    int searchindex = search(hash, searchKey) ;
    if(searchindex != -1){
        printf(" %d found in index  %d \n", searchKey,searchindex);
    }else {
        printf("Can't search - Not in the Table\n");
    }
break;

case 3 :  // ลบข้อมูลออกจากตาราง
    printf("\n\t\t Delete");
    printf("\n-------------------------------------------------\n");
    printf("Enter number to remove : ");
    scanf("%d",&removeKey);
```

```
            delete(hash,removeKey);
            printf("-----------------------------------------------\n");
            printf("\t\tAfter Delete\n");
            display(hash);
        break;

        case 4 : // ออกจากโปรแกรม
        printf("\n(Returning to main menu)\n");
        main();

        default: // เลือกตัวเลขที่ไม่มีในเมนู
        printf("\t!!Invalid choice,Please Enter again!!\n");
        break;
        }
        printf("-----------------------------------------------\n");
    }while(enter != 4);

    return 0 ;
}

if (choice==2) {
    int size, i, key;
```

```
// รับขนาดของตารางและค่า mod โดยระมัดระวัง
printf("\n\t\tLinear Probing");
printf("\n-------------------------------------------------\n");
printf("Enter size of the Table : ");
scanf("%d", &size);
while (size <= 0) {
    printf("Please input number more than zero!\n");
    printf("\nEnter size of the Table : ");
    scanf("%d", &size);
}
printf("(Size of table is %d)\n",size);



struct NodeL* hashTable[size];


for (i = 0; i < size; i++) {
    hashTable[i] = NULL;
}


while (1) {
    printf("-------------------------------------------------\n");
    printf("\nPress 1. Insert\n       2. Search\n       3. Display\n       4. Delete\n       5. Exit\n");
    printf("\nEnter number : ");
```

```
int choice;
scanf("%d", &choice);

switch (choice) {
    case 1:
        while(1) {
            printf("\n\t\tInsert\n");
            printf("-----------------------------------------------\n");
            printf("Enter value you want to insert in the table: ");
            scanf("%d", &key);
            if (key <= 0) {
                printf("\nInvalid size Please input Positive Integer\n");
                printf("-----------------------------------------------\n");
                while (getchar() != '\n');
            }else {
                while (getchar() != '\n');
                break;
        }
    }
        linearProbing(key, hashTable, size);
        break;
    case 2:
```

```
    printf("\n\t\tSearch\n");
    printf("-------------------------------------------------\n");
    printf("Enter value you want to search in the table: ");
    scanf("%d", &key);
    int searchIndex = searchL(key, hashTable, size);
    if (searchIndex != -1) {
        printf("found %d in index %d of the table\n", key, searchIndex);
    } else {
        printf("can't found %d in the table\n", key);
    }
    break;
case 3:
    printf("\n\t\tDisplay\n");
    printf("-------------------------------------------------\n");
    printf("Hashing Table\n");
    for (i = 0; i < size; i++) {
        printf("Index %d: ", i);
        struct NodeL* current = hashTable[i];
        while (current != NULL) {
            printf("%d -> ", current -> key);
            current = current->next;
        }
        printf("NULL\n");
```

```
    }
    break;
case 4:
    printf("\n\t\tDelete\n");
    printf("-----------------------------------------------\n");
    printf("Enter value you want to delete: ");
    scanf("%d", &key);
    deleteL(key, hashTable, size);
    break;

// case 5 เพื่อใช้งานฟังก์ชัน delete
case 5:
    printf("\n(Returning to main menu)\n");
    // ตรวจสอบการคืนหน่วยความจำและปิดโปรแกรม
    for (i = 0; i < size; i++) {
        struct NodeL* current = hashTable[i];
        while (current != NULL) {
            struct NodeL* temp = current;
            current = current->next;
            free(temp);
        }
    }
```

```
                main();
            }
        }


        }


        if (choice==3) {
            Quadratic();
        }
        if (choice == 4) {
            printf("-------------------------------------------------\n");
            printf("\t\tThank you ! ");
            exit;
        }
}
```

ตัวอย่างแสดงผลทางหน้าจอ

```
                Hashing Type
------------------------------------------------
1.Separate Chain
2.Linear Probing
3.Quadratic Probing
4.Exit program
------------------------------------------------
Enter number : 1

                Separate Chain
------------------------------------------------
Enter table size (must be greater than 0): 3
------------------------------------------------
                table size is 3
------------------------------------------------

Press 1. Insert
      2. Search
      3. Delete
      4. Exit

Enter your menu : 1
```

```

Press 1. Insert
      2. Search
      3. Delete
      4. Exit

Enter your menu : 1

                Insert
------------------------------------------------
Enter number of values to insert : 3
Enter value 1: 3
Enter value 2: 6
Enter value 3: 9

                 Hash Table
------------------------------------------------
Bucket[0] -> 3 -> 6 -> 9 -> NULL
Bucket[1] -> NULL
Bucket[2] -> NULL
------------------------------------------------
```

```
Press 1. Insert
      2. Search
      3. Delete
      4. Exit

Enter your menu : 2

                Search
------------------------------------------------
Enter the number to search : 4
Can't search - Not in the Table
------------------------------------------------

Press 1. Insert
      2. Search
      3. Delete
      4. Exit

Enter your menu : 3

                Delete
------------------------------------------------
Enter number to remove : 3
------------------------------------------------
               After Delete

                Hash Table
------------------------------------------------
Bucket[0] -> 6 -> 9 -> NULL
Bucket[1] -> NULL
Bucket[2] -> NULL
------------------------------------------------
```