



รายงาน

เรื่อง โปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

รายงาน

เรื่อง โปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

## คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ENGCE124 โครงสร้างข้อมูลและขั้นตอนวิธี (Data Structures and Algorithms) หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่ ในระดับปริญญาตรีปีที่ 2 โดยมีจุดประสงค์ในการอธิบายโค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) รวมถึงอธิบายหลักการทำงานของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) และอธิบายผลลัพธ์การใช้งานโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้สนใจ หรือนักศึกษาทุกท่านที่กำลังหาศึกษาในหัวข้อของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขอภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาหวงศ์

วันที่ 28/09/2567

## สารบัญ

	หน้า
คำนำ	ก
สารบัญ	๗
โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย	1
หลักการทำงานของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)	9
ผลลัพธ์การใช้งานโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)	22
บรรณานุกรม	25

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย

```
#include <stdio.h> //ใช้ printf

#include <conio.h> //ใช้ getch

#include <stdlib.h> //ใช้ random

#include <time.h> //ใช้ time

#include <windows.h> //ใช้ Sound

#define MaxData 100 // กำหนด Max Data

#define Lo 1 // กำหนดค่า Lo=1

int Data[MaxData];

int N, M, key, Addr, Times;

bool result;

struct Node // ประกาศโครงสร้างของโหนด

{

    int info;

    struct Node *link;

};

struct Node *Start[MaxData], *H1, *p; // ประกาศตัวชี้ (pointer) โหนด

Node *Allocate() // จัดสรร 1 โหนดจากพื้นที่หน่วยความจำ

{

    struct Node *temp;

    temp = (Node*)malloc(sizeof(Node)); // จัดสรรโหนดตามขนาดที่กำหนด

    return(temp);
```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

}

bool Duplicate(int i, int Data1) // ตรวจสอบข้อมูลซ้ำ
{
    int j;

    for(j = 1; j <= i; j++)
    {
        if(Data1 == Data[j])
            return(true);
    }

    return(false);
}

void PrepareRawKey(int N)
{
    int i, temp;

    srand(time(NULL)); // สร้างหมายเลขสุ่มที่แตกต่างกันใน rand()

    for (i = 0; i < N; i++)
    {
        temp = (rand() % 989) + 10; // สุ่มหมายเลขที่แตกต่างกันระหว่าง 10..999

        while(Duplicate(i - 1, temp)) // วนลูปถ้ายังมีข้อมูลซ้ำ
        {
            temp = (rand() % 989) + 10; // สุ่มใหม่อีกครั้ง
        }

        Data[i] = temp; // เก็บหมายเลขใหม่
    }
}

```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

    } // จบ for

} // จบฟังก์ชัน

void DispKey(int N)

{

    int i;

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

        printf(" %2d ", Data[i]); // แสดงข้อมูลใน Data[]

    printf("\n");

}

void CreateHead(int Head)

{

    int i;

    struct Node *p;

    for (i = 1; i <= Head; i++) // นับตามจำนวน Head

    {

        p = Allocate();

        p->info = NULL;

        p->link = NULL; // กำหนด NEXT = NULL

        Start[i] = p; // กำหนด Start ของแต่ละโหนดเป็นที่อยู่ของโหนดแรก

    } // จบ for

}

```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

void CreateHashTable(int N)

{

    int i;

    struct Node *H1, *p;

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

    {

        Addr = Data[i] % M + Lo; // คำนวณที่อยู่ของคีย์ (Addr = K mod M + Lo)

        H1 = Start[Addr];

        if(H1->info == NULL) // ถ้าเป็นโหนดหัว

            H1->info = Data[i];

        else

        {

            while(H1->link != NULL)

                H1 = H1->link;

            p = Allocate(); // เพิ่มโหนดใหม่

            p->info = Data[i];

            p->link = NULL;

            H1->link = p;

        } // จบ if

    } // จบ for

}

```



## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

void DispHashTable()
{
    int i;

    struct Node *H1;

    for(i = 1; i <= M; i++)
    {
        H1 = Start[i];

        printf("\nAddress %2d : ", i);

        while(H1 != NULL)
        {
            printf("%3d ", H1->info);

            H1 = H1->link; // ข้ามไปโหนดถัดไป
        }

        // จบ for
        printf("\n");
    }

    bool SearchHash(int key)
    {
        struct Node *H1;

        Addr = key % M + Lo; // คำนวณที่อยู่ของคีย์ (Addr = K mod M + Lo)

        H1 = Start[Addr];
    }
}

```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

Times = 0;

while(H1 != NULL)

{

    Times++; // เพิ่มตัวนับ Times

    if(H1->info == key)

        return(true); // พบแล้ว

    else

        H1 = H1->link;

}

return(false); // ไม่พบ

}

int main()

{

    printf("HASHING SEARCH(DYNAMIC CHAINING)\n");

    printf("=====\n");

    N = 32;

    M = N * 0.50; // กำหนด M = 50% ของ N

    PrepareRawKey(N);

    printf("Raw key :\n");

    DispKey(N); // แสดงคีย์ดิบ

    printf("-----\n");

```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```

CreateHead(M); // สร้างโหนดหัว

CreateHashTable(N);

while(key != -999)

{

    DispHashTable();

    printf("-----\n");

    printf("\nEnter Key for Search (-999 for EXIT) = ");

    scanf("%d", &key); // อ่านคีย์จากคีย์บอร์ด

    if(key != -999)

    {

        result = SearchHash(key);

        printf("Key Address : %d\n", Addr);

        printf("Searching Time : %d\n", Times);

        printf("Result...");

        if(result)

            printf("FOUND\n"); // ถ้าพบ

        else

        {

            Beep(600,600);

            printf("NOT FOUND!!\n"); // ถ้าไม่พบ

        }

    }

}

```

## โค้ดของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) พร้อมคำอธิบาย (ต่อ)

```
printf("----- Searching Finished\n");  
  
} // จบ if  
  
} // จบ While  
  
return(0);  
  
} // จบ Main
```

## หลักการทำงานของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

โปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) เป็นการค้นหาข้อมูลโดยใช้เทคนิค Hashing ซึ่งเป็นการแปลงข้อมูลที่เรากำลังต้องการเก็บลงในตารางแฮช โดยใช้สูตรคำนวณคีย์เพื่อลดปัญหาการชนกันของตำแหน่งที่อยู่ภายในตารางผ่านเทคนิคที่เรียกว่า Dynamic Chaining หรือการจัดเก็บข้อมูลแบบเชื่อมโยงโหนด (Linked List) หากเกิดการชนกันที่ตำแหน่งเดียวกัน โปรแกรมนี้ประกอบด้วยหลายฟังก์ชันที่ทำงานร่วมกัน เพื่อจัดการข้อมูลในตารางแฮชและค้นหาข้อมูลในตารางนั้นได้อย่างมีประสิทธิภาพ

### 1. การนำเข้าไลบรารี (Library Inclusion)

```
#include <stdio.h> //ใช้ printf
#include <conio.h> //ใช้ getch
#include <stdlib.h> //ใช้ random
#include <time.h> //ใช้ time
#include <windows.h> //ใช้ Sound
```

ในส่วนของการนำเข้าไลบรารี จะมีรายละเอียด ดังต่อไปนี้

- `#include <stdio.h>` : ไลบรารี `stdio.h` (Standard Input/Output) เป็นไลบรารีที่ให้ฟังก์ชันสำหรับการจัดการอินพุตและเอาต์พุต เช่น การแสดงผลบนหน้าจอ (ด้วย `printf()`) หรือรับข้อมูลจากผู้ใช้ (ด้วย `scanf()`)
- `#include <conio.h>` : ไลบรารี `conio.h` เป็นไลบรารีที่มักใช้ในระบบปฏิบัติการ DOS หรือ Windows รุ่นเก่า ฟังก์ชันหลักคือ `getch()` ที่รอรับอินพุตจากผู้ใช้ โดยไม่ต้องกด Enter เพื่อยืนยัน และไม่แสดงผลลัพธ์บนหน้าจอ
- `#include <stdlib.h>` : ไลบรารี `stdlib.h` (Standard Library) เป็นไลบรารีที่มีฟังก์ชันที่เกี่ยวข้องกับการจัดการหน่วยความจำแบบไดนามิก (`malloc()` สำหรับจัดสรรหน่วยความจำ และ `free()` สำหรับคืนหน่วยความจำ) รวมถึงฟังก์ชันสุ่มตัวเลข (`rand()` และ `srand()`)
- `#include <time.h>` : ไลบรารี `time.h` เป็นไลบรารีที่มีฟังก์ชัน `time()` ซึ่งใช้สำหรับการตั้งค่า seed เพื่อให้การสุ่มตัวเลขใน `rand()` แตกต่างกันทุกครั้งที่รันโปรแกรม

- `#include <windows.h>` : ไลบรารี `windows.h` เป็นไลบรารีเฉพาะของระบบปฏิบัติการ Windows ใช้ฟังก์ชันที่เกี่ยวข้องกับระบบ Windows เช่น `Beep()` ซึ่งส่งเสียงเตือนเมื่อเกิดเงื่อนไขที่ต้องแจ้งให้ผู้ใช้ทราบ (เช่น ค้นหาข้อมูลไม่พบ)

## 2. การประกาศค่าคงที่ (Constant Definitions)

```
#define MaxData 100 // กำหนด Max Data
#define Lo 1 // กำหนดค่า Lo=1
```

ในส่วนของการประกาศค่าคงที่ จะมีรายละเอียด ดังต่อไปนี้

- `#define MaxData 100` : คำสั่งนี้กำหนดให้ `MaxData` มีค่าเท่ากับ 100 ซึ่งหมายถึงโปรแกรมนี้สามารถเก็บข้อมูลได้สูงสุด 100 ชุดในอาร์เรย์ `Data[]`. ตัวเลขนี้ถูกกำหนดแบบค่าคงที่ (constant) เพื่อไม่ให้เปลี่ยนแปลงระหว่างการทำงานของโปรแกรม
- `#define Lo 1` : กำหนดค่าคงที่ `Lo` เป็น 1 ซึ่งมีบทบาทสำคัญในขั้นตอนการคำนวณที่อยู่ (Address) ในตารางแฮชด้วยสูตร  $Addr = K \% M + Lo$ . ค่านี้ช่วยให้ผลลัพธ์ของการคำนวณ Address เริ่มที่ 1 แทนที่จะเป็น 0 (ซึ่งเป็นค่าเริ่มต้นใน C)

## 3. การประกาศตัวแปรทั่วไป (Variable Declarations)

```
int Data[MaxData];

int N, M, key, Addr, Times;

bool result;
```

ในส่วนของการประกาศตัวแปรทั่วไปจะมีรายละเอียด ดังต่อไปนี้

- `Data[MaxData]` : อาร์เรย์นี้มีขนาด 100 ตัว ใช้เก็บข้อมูลที่สุ่มขึ้นมา โปรแกรมจะจัดเก็บข้อมูลในรูปแบบของจำนวนเต็ม (integer) ซึ่งแต่ละตัวจะถูกจัดเก็บในแต่ละช่องของอาร์เรย์
- `N` : ใช้เก็บจำนวนข้อมูลที่ใช้จริงในการรันโปรแกรม โดยในตัวอย่าง `N` ถูกกำหนดเป็น 32 (ใน `main()`) หมายความว่าโปรแกรมจะสุ่มข้อมูล 32 ชุดจาก 100 ช่องของอาร์เรย์
- `M` : ตัวแปรนี้เก็บจำนวนตำแหน่ง (slot) ในตารางแฮช ซึ่งถูกกำหนดให้เป็น 50% ของ `N` ( $M = 0.5 * N$ ) การกำหนด `M` เป็นครึ่งหนึ่งของ `N` เป็นการออกแบบให้ลดการใช้หน่วยความจำ
- `Key` : ใช้เก็บค่าคีย์ที่ผู้ใช้ป้อนเข้ามาเพื่อค้นหาในตารางแฮช

- Addr : เก็บค่าที่อยู่ (address) ในตารางแฮชที่คำนวณได้จากคีย์โดยใช้สูตร  $Addr = key \% M + Lo$
- Times : ตัวแปรที่นับจำนวนครั้งที่ใช้ในการค้นหาข้อมูล โดยเริ่มจากศูนย์และเพิ่มขึ้นทุกครั้งที่ต้องข้ามไปยังโหนดถัดไปใน Linked List
- result : ตัวแปรชนิด bool ใช้เก็บผลลัพธ์ของการค้นหาในตารางแฮช ถ้าค้นพบข้อมูลจะเป็น true ถ้าไม่พบจะเป็น false

#### 4. การประกาศโครงสร้างข้อมูล (Struct Declarations)

```
struct Node // ประกาศโครงสร้างของโหนด
{
    int info;

    struct Node *link;
};
```

ในส่วนของการประกาศโครงสร้างข้อมูล จะมีรายละเอียด ดังต่อไปนี้

- struct Node : เป็นโครงสร้างข้อมูลที่ใช้ในการสร้าง Linked List ซึ่งมี 2 ส่วน ประกอบไปด้วย
  - info : เก็บข้อมูลที่เป็นจำนวนเต็ม (integer) ซึ่งในโปรแกรมนี้จะเก็บคีย์ที่สุ่มขึ้นมา
  - link : ตัวชี้ (pointer) ที่ชี้ไปยังโหนดถัดไปใน Linked List ถ้าไม่มีโหนดถัดไป link จะชี้ไปที่ NULL

โดยสาเหตุที่ใช้งาน Linked List เพื่อใช้ในการแก้ปัญหาการชนกันของข้อมูล (collision) ในตารางแฮช เมื่อมีการชนกัน (ข้อมูลหลายตัวมีค่า Address เดียวกัน) ข้อมูลเหล่านั้นจะถูกเก็บใน Linked List ที่ตำแหน่งนั้น

#### 5. การประกาศตัวชี้ (Pointer Declarations)

```
struct Node *Start[MaxData], *H1, *p; // ประกาศตัวชี้ (pointer) โหนด
```

ในส่วนของการประกาศตัวชี้ จะมีรายละเอียด ดังต่อไปนี้

- Start[MaxData] : อาร์เรย์ของตัวชี้ (pointer) ที่ใช้เก็บตำแหน่งหัวของ Linked List ในแต่ละช่องของตารางแฮช. แต่ละช่องในอาร์เรย์ Start[] จะเก็บตัวชี้ไปยังโหนดแรกของ Linked List ที่ตำแหน่งนั้น

- H1 : ตัวชี้ที่ใช้ในการไล่ดูข้อมูลใน Linked List ของตำแหน่งนั้นๆ. เมื่อโปรแกรมค้นหาข้อมูล มันจะเริ่มจากโหนดแรก (ที่ชี้โดย Start[]) และใช้ H1 ในการไล่ดูโหนดถัดๆ ไปจนกว่าจะพบคีย์หรือถึงจุดสิ้นสุดของ Linked List
- p : ตัวชี้ที่ใช้ในการสร้างโหนดใหม่ใน Linked List. เมื่อมีการเพิ่มข้อมูลใหม่ลงในตารางแฮช (ในกรณีเกิดการชนกัน) p จะถูกใช้เพื่อจัดสรรหน่วยความจำใหม่สำหรับโหนดนั้น

## 6. ฟังก์ชัน Allocate

```
Node *Allocate() // จัดสรร 1 โหนดจากพื้นที่หน่วยความจำ
{
    struct Node *temp;

    temp = (Node*)malloc(sizeof(Node)); // จัดสรรโหนดตามขนาดที่กำหนด

    return(temp);
}
```

ฟังก์ชัน Allocate ทำหน้าที่จัดสรรหน่วยความจำสำหรับการสร้าง Node ใหม่ ซึ่งเป็นโครงสร้างข้อมูลที่ใช้ในการเก็บข้อมูลในรูปแบบ Linked List เมื่อมีการเพิ่มข้อมูลใหม่ลงในตารางแฮช ฟังก์ชันนี้จะคืนค่าหน่วยความจำที่จัดสรรแล้วให้โปรแกรมเพื่อนำไปใช้งานต่อได้ทันที โดยหลักการทำงานเริ่มต้นโดยการจองพื้นที่หน่วยความจำสำหรับ Node ใหม่โดยใช้ฟังก์ชัน malloc() จากไลบรารี <stdlib.h> โดยโครงสร้างของ Node ประกอบด้วย 2 ส่วนคือ info ที่ใช้เก็บข้อมูลที่จะแฮชและจัดเก็บ และ link ที่ใช้เป็นตัวชี้ไปยังโหนดถัดไปใน Linked List โดยหน่วยความจำที่ถูกจัดสรรจะคืนค่าเป็นที่อยู่ของโหนดนั้น ซึ่งสามารถนำไปใช้ในโครงสร้างแบบ Dynamic Chaining

## 7. ฟังก์ชัน Duplicate

```
bool Duplicate(int i, int Data1) // ตรวจสอบข้อมูลซ้ำ
{
    int j;
```



## 7. ฟังก์ชัน Duplicate (ต่อ)

```

for(j = 1; j <= i; j++)

{

    if(Data1 == Data[j])

        return(true);

}

return(false);

}

```

ฟังก์ชัน Duplicate ใช้ในการตรวจสอบว่าข้อมูลที่สร้างขึ้นใหม่มีความซ้ำซ้อนกับข้อมูลก่อนหน้าในอาร์เรย์หรือไม่ ถ้ามีการซ้ำซ้อน ฟังก์ชันจะคืนค่าเป็น true เพื่อแจ้งว่าไม่ควรใช้ข้อมูลนี้ โดยหลักการทำงานเริ่มต้นโดยการใช้ลูปเพื่อตรวจสอบข้อมูลตั้งแต่ตำแหน่งแรกจนถึงข้อมูลที่กำลังถูกเพิ่ม (i) เพื่อหาว่าข้อมูลที่ได้รับ (Data1) ซ้ำกับข้อมูลก่อนหน้าหรือไม่ ถ้าพบข้อมูลซ้ำ ฟังก์ชันจะคืนค่า true เพื่อให้โปรแกรมทราบว่าควรสร้างข้อมูลใหม่โดยใช้การสุ่มใหม่ แต่ถ้าไม่ซ้ำ จะคืนค่า false ซึ่งหมายความว่าสามารถใช้ข้อมูลนี้ได้

## 8. ฟังก์ชัน PrepareRawKey

```

void PrepareRawKey(int N)

{

    int i, temp;

    srand(time(NULL)); // สร้างหมายเลขสุ่มที่แตกต่างกันใน rand()

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

    {

        temp = (rand() % 989) + 10; // สุ่มหมายเลขที่แตกต่างกันระหว่าง 10..999

        while(Duplicate(i - 1, temp)) // วนลูปถ้ายังมีข้อมูลซ้ำ

            temp = (rand() % 989) + 10; // สุ่มใหม่อีกครั้ง
    }
}

```

## 8. ฟังก์ชัน PrepareRawKey (ต่อ)

```

        Data[i] = temp; // เก็บหมายเลขใหม่

    } // จบ for

} // จบฟังก์ชัน

```

ฟังก์ชัน PrepareRawKey ใช้ในการสร้างข้อมูลแบบสุ่มที่ไม่ซ้ำกัน โดยจะนำข้อมูลที่สร้างได้ไปเก็บในอาร์เรย์ Data[] ซึ่งจะใช้ในตารางแฮชต่อไป โดยหลักการทำงานเริ่มต้นโดยใช้ฟังก์ชัน srand() เพื่อเริ่มต้นค่าการสุ่มด้วยการใช้ค่าเวลา (time(NULL)) ทำให้การสุ่มข้อมูลแตกต่างกันทุกครั้งที่โปรแกรมทำงาน จากนั้นจะใช้ rand() เพื่อสุ่มหมายเลขในช่วง 10 ถึง 99 หลังจากสุ่มได้ค่า จะตรวจสอบความซ้ำซ้อนของข้อมูลโดยใช้ฟังก์ชัน Duplicate() ถ้าข้อมูลซ้ำ จะทำการสุ่มหมายเลขใหม่จนกว่าจะได้ข้อมูลที่ไม่ซ้ำ

## 9. ฟังก์ชัน DispKey

```

void DispKey(int N)

{

    int i;

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

        printf(" %2d ", Data[i]); // แสดงข้อมูลใน Data[]

    printf("\n");

}

```

ฟังก์ชัน DispKey ใช้แสดงค่าข้อมูลในอาร์เรย์ Data[] ซึ่งเก็บข้อมูลที่สร้างได้จากฟังก์ชัน PrepareRawKey() ให้ผู้ใช้ดู โดยหลักการทำงานเริ่มต้นโดยการใช้ลูป for เพื่อแสดงข้อมูลในอาร์เรย์ Data[] โดยแสดงผลข้อมูลแต่ละตัวแบบเว้นระยะเพื่อให้อ่านง่าย โดยข้อมูลที่แสดงจะเป็นผลลัพธ์จากการสุ่มในฟังก์ชัน PrepareRawKey()

## 10. ฟังก์ชัน CreateHead

```

void CreateHead(int Head)
{
    int i;

    struct Node *p;

    for (i = 1; i <= Head; i++) // นับตามจำนวน Head
    {
        p = Allocate();

        p->info = NULL;

        p->link = NULL; // กำหนด NEXT = NULL

        Start[i] = p; // กำหนด Start ของแต่ละโหนดเป็นที่อยู่ของโหนดแรก
    } // จบ for
}

```

ฟังก์ชัน CreateHead ทำหน้าที่สร้าง Head Node สำหรับการจัดเก็บข้อมูลในรูปแบบ Linked List โดยฟังก์ชันจะจัดสรรหน่วยความจำให้กับโหนดหัวแต่ละตัวและกำหนดค่าเริ่มต้นให้กับแต่ละโหนด โดยหลักการทำงานเริ่มต้นโดยการสร้างโหนดหัวที่ว่างเปล่าสำหรับแต่ละตำแหน่งในตารางแฮช ซึ่งโหนดหัวจะเป็นตัวเริ่มต้นของ Linked List ในแต่ละตำแหน่ง จากนั้นฟังก์ชัน Allocate() จะถูกเรียกเพื่อจัดสรรหน่วยความจำสำหรับโหนดหัว และโหนดเหล่านี้จะถูกเก็บในอาร์เรย์ Start[]

## 11. ฟังก์ชัน CreateHashTable

```

void CreateHashTable(int N)
{
    int i;

    struct Node *H1, *p;

```

## 11. ฟังก์ชัน CreateHashTable (ต่อ)

```

for(i = 0; i < N; i++)
{
    Addr = Data[i] % M + Lo; // คำนวณที่อยู่ของคีย์ (Addr = K mod M + Lo)

    H1 = Start[Addr];

    if(H1->info == NULL) // ถ้าเป็นโหนดหัว
        H1->info = Data[i];

    else
    {
        while(H1->link != NULL)

            H1 = H1->link;

        p = Allocate(); // เพิ่มโหนดใหม่

        p->info = Data[i];

        p->link = NULL;

        H1->link = p;

    } // จบ if

} // จบ for
}

```

ฟังก์ชัน CreateHashTable ทำหน้าที่สร้างตารางแฮชโดยคำนวณที่อยู่ของข้อมูลจากคีย์ที่ได้จากอาร์เรย์ Data[] โดยใช้สูตร  $Addr = K \bmod M + Lo$  เพื่อหาตำแหน่งจัดเก็บในตาราง หากตำแหน่งเกิดการชนกัน ฟังก์ชันจะเพิ่มข้อมูลลงใน Linked List ที่ตำแหน่งนั้น โดยหลักการทำงานของฟังก์ชันนี้ใช้การคำนวณแฮช  $Addr = K \bmod M + Lo$  เพื่อหาตำแหน่งการจัดเก็บข้อมูลในตารางแฮช ถ้าตำแหน่งที่คำนวณได้ยังว่างอยู่ จะเก็บ

ข้อมูลในโหนดหัวของตำแหน่งนั้นทันที แต่ถ้าตำแหน่งที่คำนวณได้มีข้อมูลอยู่แล้ว (การชนกัน) จะเพิ่มข้อมูลใหม่ในรูปแบบ Linked List โดยสร้างโหนดใหม่และเชื่อมต่อกับโหนดเดิม

## 12. ฟังก์ชัน DispHashTable

```
void DispHashTable()
{
    int i;

    struct Node *H1;

    for(i = 1; i <= M; i++)
    {
        H1 = Start[i];

        printf("\nAddress %2d : ", i);

        while(H1 != NULL)
        {
            printf("%3d ", H1->info);

            H1 = H1->link; // ข้ามไปโหนดถัดไป
        }

        // จบ for

        printf("\n");
    }
}
```

ฟังก์ชัน DispHashTable ใช้แสดงตารางแฮชที่สร้างขึ้นพร้อมกับข้อมูลที่ถูกจัดเก็บในแต่ละตำแหน่งของตาราง หากมีข้อมูลมากกว่าหนึ่งในตำแหน่งเดียวกัน ฟังก์ชันจะแสดงข้อมูลเหล่านั้นด้วยการแสดง Linked List โดยหลักการทำงานของฟังก์ชันนี้จะใช้ลูปเพื่อแสดงข้อมูลในตารางแฮชโดยแสดงที่อยู่แต่ละตำแหน่งและข้อมูลที่จัดเก็บ

ในรูปแบบ Linked List โดยข้อมูลในแต่ละตำแหน่งจะแสดงผลแบบเป็นลำดับโดยเรียงจากโหนดหัวไปจนถึงโหนดสุดท้าย

### 13. ฟังก์ชัน SearchHash

```
bool SearchHash(int key)
{
    struct Node *H1;

    Addr = key % M + Lo; // คำนวณที่อยู่ของคีย์ (Addr = K mod M + Lo)

    H1 = Start[Addr];

    Times = 0;

    while(H1 != NULL)
    {
        Times++; // เพิ่มตัวนับ Times

        if(H1->info == key)

            return(true); // พบแล้ว

        else

            H1 = H1->link;

    }

    return(false); // ไม่พบ
}
```

ฟังก์ชัน SearchHash ใช้สำหรับการค้นหาข้อมูลในตารางแฮช โดยการคำนวณที่อยู่จากคีย์ที่รับเข้ามา หากพบข้อมูล ฟังก์ชันจะคืนค่า true เพื่อระบุว่าพบข้อมูลแล้ว มิฉะนั้นจะคืนค่า false โดยหลักการทำงานของฟังก์ชันนี้คำนวณตำแหน่งในตารางแฮชโดยใช้สูตร  $Addr = key \% M + Lo$  ซึ่งคีย์ที่ต้องการค้นหาจะถูกหารเอาเศษด้วยค่า M (จำนวนตำแหน่งในตารางแฮช) จากนั้นบวกด้วย Lo ซึ่งช่วยจัดการการกระจายของคีย์ในตาราง

จากนั้นจะใช้รูปเพื่อค้นหาคีย์ใน Linked List ที่ตำแหน่งนั้น ถ้าพบคีย์ ฟังก์ชันจะคืนค่า true แต่ถ้าไม่พบก็จะคืนค่า false โดยจำนวนครั้งในการค้นหา (Times) จะถูกนับเพื่อแสดงให้เห็นถึงความซับซ้อนของการค้นหาในแต่ละครั้ง

#### 14. ฟังก์ชัน main

```
int main()
{
    printf("HASHING SEARCH(DYNAMIC CHAINING)\n");

    printf("=====\n");

    N = 32;

    M = N * 0.50; // กำหนด M = 50% ของ N

    PrepareRawKey(N);

    printf("Raw key :\n");

    DispKey(N); // แสดงคีย์ดิบ

    printf("-----\n");

    CreateHead(M); // สร้างโหนดหัว

    CreateHashTable(N);

    while(key != -999)
    {
        DispHashTable();

        printf("-----\n");

        printf("\nEnter Key for Search (-999 for EXIT) = ");

        scanf("%d", &key); // อ่านคีย์จากคีย์บอร์ด

        if(key != -999)
```

## 14. ฟังก์ชัน main (ต่อ)

```

{

    result = SearchHash(key);

    printf("Key Address : %d\n", Addr);

    printf("Searching Time : %d\n", Times);

    printf("Result...");

    if(result)

        printf("FOUND\n"); // ถ้าพบ

    else

    {

        Beep(600,600);

        printf("NOT FOUND!!\n"); // ถ้าไม่พบ

    }

    printf("----- Searching Finished\n");

} // จบ if

} // จบ While

return(0);

} // จบ Main

```

ฟังก์ชัน main เป็นฟังก์ชันหลักของโปรแกรม ทำหน้าที่เรียกใช้ฟังก์ชันอื่นๆ เพื่อดำเนินการสร้างข้อมูลดิบ สร้างตารางแฮช และอนุญาตให้ผู้ใช้ค้นหาคีย์ที่ต้องการในตารางแฮช โดยหลักการทำงานเริ่มต้นจากการสุ่มข้อมูลและสร้างตารางแฮช โดยโปรแกรมจะรอรับการป้อนคีย์จากผู้ใช้เพื่อตรวจสอบว่าคีย์นั้นมีอยู่ในตารางแฮชหรือไม่ โดยเมื่อค้นหาพบจะคืนค่าพร้อมทั้งแสดงตำแหน่งและจำนวนครั้งที่ใช้ในการค้นหา ซึ่งผู้ใช้สามารถ



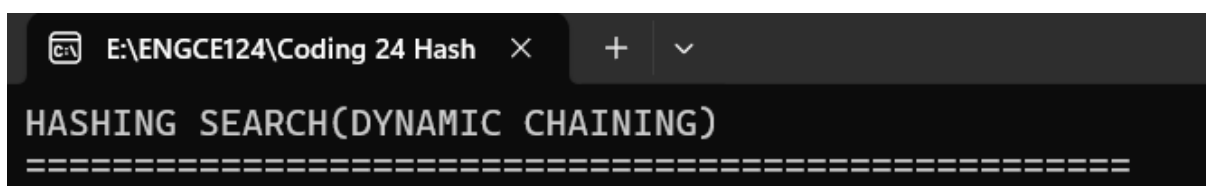
ป้อนคีย์เพื่อค้นหาเรื่อยๆ จนกว่าจะป้อน -999 เพื่อหยุดการทำงานของโปรแกรม สุดท้ายเมื่อคีย์ไม่ถูกค้นพบ ระบบจะส่งเสียงเตือนโดยใช้ฟังก์ชัน Beep(600,600) จากไลบรารี <windows.h>

## ผลลัพธ์การใช้งานโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

โปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD) เป็นโปรแกรมสำหรับการจัดการข้อมูลด้วยการ Hashing Search โดยใช้วิธีการจัดการการชนกัน (collision) แบบ Dynamic Chaining ซึ่งเป็นการใช้ Linked List เพื่อแก้ปัญหาการชนกันของข้อมูลในตารางแฮช

### 1. การเริ่มต้นของโปรแกรม

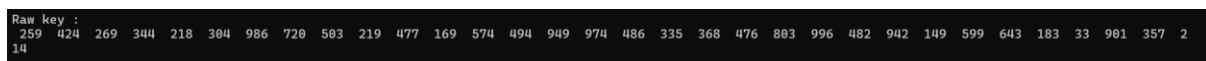
เมื่อรันโปรแกรม ผู้ใช้จะเห็นข้อความหัวข้อของโปรแกรมที่มีชื่อว่า “BINARY SEARCH” แสดงออกมาบนหน้าจอคอนโซล



```
E:\ENGCE124\Coding 24 Hash  X  +  v
=====
HASHING SEARCH(DYNAMIC CHAINING)
=====
```

### 2. การสุ่มข้อมูลและแสดงผลข้อมูล

เมื่อเริ่มต้นโปรแกรม ฟังก์ชัน PrepareRawKey() จะทำการสุ่มตัวเลขขึ้นมา 32 ค่า (กำหนดในตัวแปร N ว่าเป็น 32) โดยค่าที่สุ่มจะอยู่ในช่วงระหว่าง 10 ถึง 999 และจะเก็บค่าที่สุ่มได้ลงในอาร์เรย์ Data[] นอกจากนี้ โปรแกรมจะตรวจสอบว่าไม่มีค่าซ้ำกันในข้อมูลที่สุ่มขึ้นมา โดย ข้อมูลที่สุ่มได้จะแสดงเป็นแถวเดียว โดยโปรแกรมจะแสดงข้อมูลที่อยู่ในอาร์เรย์ Data[] ทั้งหมด 32 ตัว



```
Raw key :
259 424 269 344 218 304 986 720 503 219 477 169 574 494 949 974 486 335 368 476 803 996 482 942 149 599 643 183 33 901 357 2
14
```

### 3. การสร้างและแสดงผลตารางแฮช (Hash Table)

โปรแกรมจะสร้างตารางแฮชขนาด 50% ของจำนวนข้อมูล ( $M = N * 0.50$  ซึ่งเท่ากับ 16) จากนั้นจะคำนวณแฮช (ที่อยู่) สำหรับข้อมูลแต่ละตัวในตารางแฮชด้วยสูตร  $Addr = K \% M + Lo$  (โดยที่ K คือข้อมูล และ Lo คือค่าคงที่ 1) โดยถ้าช่องในตารางแฮชว่าง โปรแกรมจะเก็บข้อมูลไว้ที่ช่องนั้น แต่ถ้าช่องนั้นมีข้อมูลอยู่แล้ว (เกิดการชนกัน) โปรแกรมจะสร้าง Linked List ในช่องนั้น โดยโหนดแรกจะเป็นข้อมูลที่ชนกัน และข้อมูลใหม่จะถูกต่อท้ายในลิงก์ลิสต์ ซึ่งหลังจากการคำนวณแฮชและสร้างตารางแฮชเสร็จสิ้น โปรแกรมจะแสดงผลตารางแฮชในรูปแบบของแฮชและข้อมูลที่จัดเก็บในแต่ละแฮช. ถ้ามีข้อมูลหลายตัวในแฮชเดียวกัน (การชนกัน) ข้อมูลเหล่านั้นจะแสดงในรูปแบบของลิงก์ลิสต์ ในการแสดงผลตารางแฮชแสดงแฮชตั้งแต่ 1 ถึง M (ซึ่งคือ 16 ในตัวอย่างนี้) โดยแต่ละแฮชจะมีข้อมูลที่ถูกรวบรวมไว้ หากแฮชมีข้อมูลมากกว่าหนึ่งตัว (เกิดการชนกัน) ข้อมูลจะถูกจัดเก็บในลิงก์ลิสต์ และจะแสดงเป็นข้อมูลต่อกันไปเรื่อยๆ

```

Address 1 : 304 720 368
Address 2 : 33
Address 3 : 482
Address 4 : 259 803 643
Address 5 : 996
Address 6 : 949 149 901 357
Address 7 : 486 214
Address 8 : 503 599 183
Address 9 : 424 344
Address 10 : 169
Address 11 : 218 986
Address 12 : 219
Address 13 : 476
Address 14 : 269 477
Address 15 : 574 494 974 942
Address 16 : 335
-----

```

#### 4. การค้นหาข้อมูล

ผู้ใช้สามารถป้อนค่า คีย์ (Key) ที่ต้องการค้นหาในตารางแฮชผ่านทางคีย์บอร์ด โดยโปรแกรมจะค้นหา คีย์ในแอดเดรสที่คำนวณได้จากสูตร  $Addr = Key \% M + Lo$ . เมื่อโปรแกรมพบคีย์ที่ต้องการในลิงก์ลิสต์ โปรแกรมจะแสดงผลว่า ค้นพบ (FOUND) พร้อมกับแอดเดรสและจำนวนครั้งที่ใช้ในการค้นหา โดยในการแสดงผล โปรแกรมจะแสดงแอดเดรสของคีย์, จำนวนครั้งที่ใช้ในการค้นหา, และผลลัพธ์ของการค้นหา ถ้าคีย์ นั้นมีอยู่ในตารางแฮช จะขึ้นข้อความ FOUND และถ้าค้นหาไม่พบ โปรแกรมจะส่งเสียง Beep และขึ้นข้อความ NOT FOUND

```

Enter Key for Search(-999 for EXIT) = 269
Key Address : 14
Searching Time : 1
Result...FOUND
-----Searching Finished

Address 1 : 304 720 368
Address 2 : 33
Address 3 : 482
Address 4 : 259 803 643
Address 5 : 996
Address 6 : 949 149 901 357
Address 7 : 486 214
Address 8 : 503 599 183
Address 9 : 424 344
Address 10 : 169
Address 11 : 218 986
Address 12 : 219
Address 13 : 476
Address 14 : 269 477
Address 15 : 574 494 974 942
Address 16 : 335
-----

```

```

Enter Key for Search(-999 for EXIT) = 1000
Key Address : 9
Searching Time : 2
Result...NOT FOUND!!
-----Searching Finished

Address 1 : 304 720 368
Address 2 : 33
Address 3 : 482
Address 4 : 259 803 643
Address 5 : 996
Address 6 : 949 149 901 357
Address 7 : 486 214
Address 8 : 503 599 183
Address 9 : 424 344
Address 10 : 169
Address 11 : 218 986
Address 12 : 219
Address 13 : 476
Address 14 : 269 477
Address 15 : 574 494 974 942
Address 16 : 335
-----

```

#### 5. การหยุดการทำงานของโปรแกรม

โปรแกรมจะให้ผู้ใช้ค้นหาข้อมูลในตารางแฮชได้เรื่อยๆ จนกว่าผู้ใช้จะป้อนค่า -999 เพื่อหยุดการทำงานของโปรแกรม. เมื่อผู้ใช้ป้อน -999 โปรแกรมจะออกจากลูปการค้นหาและสิ้นสุดการทำงาน

```

Enter Key for Search(-999 for EXIT) = -999
-----
Process exited after 831.5 seconds with return value 0
Press any key to continue . . . |

```

#### ภาพรวมผลลัพธ์ของโปรแกรม HASHING SEARCH (DYNAMIC CHAINING METHOD)

```

E:\ENGCE124\Coding 24 Hash  x  +  v
HASHING SEARCH(DYNAMIC CHAINING)
=====
Raw key :
107 250 203 777 244 151 899 545 314 100 98 424 452 446 15 201 419 639 295 678 87 892 280 55 747 729 691 928 278 478 91 134
-----
Address 1 : 928
Address 2 : 545
Address 3 : 98
Address 4 : 899 419 691
Address 5 : 244 100 452
Address 6 : 0
Address 7 : 678 278 134
Address 8 : 151 295 87 55
Address 9 : 424 280
Address 10 : 777 201 729
Address 11 : 250 314
Address 12 : 107 203 747 91
Address 13 : 892
Address 14 : 0
Address 15 : 446 478
Address 16 : 15 639
-----
Enter Key for Search(-999 for EXIT) = |

```

### บรรณานุกรม

ChatGPT. ( - ). Efficient Hashing Search Using Dynamic Chaining Method. สืบค้น 28 กันยายน 2567,  
จาก <https://chatgpt.com/>