



รายงาน

เรื่อง โปรแกรม RADIX SORT

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 RADIX SORT

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 RADIX SORT รวมถึงอธิบายหลักการทำงานของโปรแกรม RADIX SORT และอธิบายผลลัพธ์การใช้งานโปรแกรม RADIX SORT

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้สนใจ หรือนักศึกษาทุกท่านที่กำลังหาศึกษาในหัวข้อของโปรแกรม RADIX SORT หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขออภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 22/09/2567

## สารบัญ

	หน้า
คำนำ	ก
สารบัญ	๗
โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย	1
หลักการทำงานของโปรแกรม RADIX SORT	6
ผลลัพธ์การใช้งานโปรแกรม RADIX SORT	15
บรรณานุกรม	17

## โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย

```
#include <stdio.h> //ใช้ printf

#include <conio.h> //ใช้ getch

#include <stdlib.h> //ใช้ random

#include <time.h> //ใช้ time

#define MaxData 100 // กำหนดค่าข้อมูลสูงสุด

#define MaxRow 10 //0..9 ในฐานสิบ

#define MaxCol 20 //0..19 ในฐานสิบ

int Data[MaxData];

int Radix[MaxRow][MaxCol]; //Radix เป็นถึงเก็บชั่วคราว ขนาดคือ [0..MaxRow, 0..MaxCol]

int N, N1;

void ClearStackPT() //ล้างทุกบล็อกล็อกเป็น 0 และใช้ Radix[0] สำหรับ SP
{
    int i;

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

        Radix[i][0] = NULL; //ตำแหน่งนี้เก็บ SP
}

void PrepareRawData(int N2)
{
    int i;

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

    for (i = 0; i < N2; i++)
```

## โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย (ต่อ)

```

        Data[i] = (rand() % 899) + 100; //สุ่มตัวเลขที่ต่างกันในช่วง 100..999
    }

void DispData(int N2) //แสดงข้อมูลในอาร์เรย์
{
    int i;

    for(i = 0; i < N2; i++)

        printf("%3d ", Data[i]);

    printf("\n");
}

void Push(int Rad, int Dat) //ใส่ข้อมูลลงใน Stack แบบขนาน โดยเก็บ SP ที่ (Rad, 0)
{
    int SP;

    SP = Radix[Rad][0] + 1; //ข้าม SP

    Radix[Rad][0] = SP; //เก็บ SP ใหม่

    Radix[Rad][SP] = Dat; //ดันข้อมูลลงไป Radix นั้น
}

void ReadStack() //อ่านข้อมูลจากแต่ละ Stack และถ่ายโอนไปยัง Data[]
{
    int i, j, k, SP;

    k = 0; //เริ่มที่ Data[] ตำแหน่งแรกคือ 0

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

```

### โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย (ต่อ)

```
{
    SP = Radix[i][0]; //ค่าจำนวนสูงสุดของแต่ละ Radix เก็บที่คอลัมน์ 0

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

    {
        Data[k] = Radix[i][j]; //ถ่ายโอนข้อมูลจากทุก Stack ไปยัง Data[]

        k++;
    }

} //จบ for

} //จบฟังก์ชัน

void RadixSort(int N2)
{
    int Digit, i, RadixNo;

    char Txt[2];

    for(Digit = 2; Digit >= 0; Digit--) //นับถอยหลัง

    {
        printf("[Digit : %d]==>\n", 3 - Digit);

        for(i = 0; i < N2; i++) //ตัวนับ Data[]

        {
            itoa(Data[i], Txt, 10); //แปลง Integer เป็น Text [itoa(อินพุต, เอาท์พุต, ฐาน)]

            RadixNo = Txt[Digit] - 48; //แปลง Text ของหลักนั้นให้เป็นตัวเลข

            Push(RadixNo, Data[i]); //ดันข้อมูลไปยัง Stack ใน Radix นั้น
        }
    }
}
```

## โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย (ต่อ)

```

    } //จบ for

    ReadStack();

    DispData(N2);

    ClearStackPT();

    } //จบ for
} //จบฟังก์ชัน

int main()
{
    printf("ASCENDING RADIX SORT\n");

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

    N = 16; //เปลี่ยนจำนวน N ที่นี่

    N1 = N; //เก็บค่า N ไปยัง N1 เพราะ N จะเป็น 0 เมื่อเสร็จสิ้น RadixSort()

    PrepareRawData(N);

    printf("Raw Data...\\n");

    DispData(N);

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

    printf("Processing Data...\\n");

    RadixSort(N);

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

    printf("Sorted Data : \\n");

```



### โค้ดของโปรแกรม RADIX SORT พร้อมคำอธิบาย (ต่อ)

```
DispData(N1); //ข้อมูลที่จัดเรียงแล้ว  
  
getch();  
  
return(0);  
  
} //จบ Main
```

## หลักการการทำงานของโปรแกรม RADIX SORT

โปรแกรม RADIX SORT เป็นตัวอย่างของการใช้อัลกอริทึม Radix Sort ซึ่งเป็นหนึ่งในวิธีการจัดเรียงข้อมูลแบบไม่ใช้การเปรียบเทียบ (Non-Comparison Sorting Algorithm) โดยอัลกอริทึมนี้จะจัดเรียงตัวเลขจากหลักหน่วยไปยังหลักสูงกว่า เช่น หลักสิบหรือหลักร้อย ซึ่งโปรแกรมนี้ได้รับการออกแบบให้จัดเรียงตัวเลขในช่วง 100 ถึง 999 ผ่านการแยกจัดเรียงทีละหลัก

### 1. การประกาศและใช้งานไลบรารี

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

ในส่วนการการประกาศและใช้งานไลบรารี มีรายละเอียดดังนี้

- `#include <stdio.h>` : ไลบรารีมาตรฐานสำหรับการจัดการอินพุต/เอาต์พุต (Input/Output) โดยเฉพาะฟังก์ชัน `printf()` ที่ใช้ในการแสดงผลข้อความหรือข้อมูลต่าง ๆ บนหน้าจอ โดยโปรแกรมนี้ใช้ `printf()` เพื่อแสดงข้อมูลและขั้นตอนต่าง ๆ ในการจัดเรียงข้อมูล
- `#include <conio.h>` : ไลบรารีที่ใช้สำหรับการรับอินพุตจากคีย์บอร์ดและควบคุมการทำงานของคอนโซล (`getch()`) เป็นฟังก์ชันที่ใช้เพื่อหยุดการทำงานของโปรแกรมและรอให้ผู้ใช้กดปุ่มใด ๆ เพื่อดำเนินการต่อ)
- `#include <stdlib.h>` : ไลบรารีนี้มีฟังก์ชันที่ใช้ในการจัดการหน่วยความจำและการสุ่มตัวเลข เช่น `rand()` สำหรับการสุ่มตัวเลข และ `srand()` สำหรับการกำหนด Seed สำหรับการสุ่ม โดยในโปรแกรมนี้ใช้สำหรับการสุ่มตัวเลขเพื่อสร้างข้อมูลดิบ (Raw Data) สำหรับการจัดเรียง
- `#include <time.h>` : ไลบรารีนี้ใช้สำหรับการทำงานที่เกี่ยวข้องกับเวลา เช่น `time(NULL)` ที่ใช้ในฟังก์ชัน `srand()` เพื่อให้โปรแกรมสุ่มตัวเลขที่แตกต่างกันในแต่ละครั้งที่รันโปรแกรม

## 2. การประกาศค่าคงที่ (Constant Definitions)

```
#define MaxData 100 // กำหนดค่าข้อมูลสูงสุด
#define MaxRow 10 //0..9 ในฐานสิบ
#define MaxCol 20 //0..19 ในฐานสิบ
```

ในส่วนการประกาศค่าคงที่ มีรายละเอียดดังนี้

- #define MaxData 100 : กำหนดค่าคงที่ MaxData ซึ่งหมายถึงจำนวนข้อมูลสูงสุดที่โปรแกรมสามารถจัดการได้ โดยในโปรแกรมนี้นำหนดให้ MaxData เท่ากับ 100 นั้นหมายความว่า array Data[] สามารถเก็บข้อมูลได้มากที่สุด 100 ตัวเลข
- #define MaxRow 10 : กำหนดจำนวนแถว (Row) ใน array Radix ซึ่งในที่นี้กำหนดเป็น 10 เนื่องจากในการใช้ Radix Sort สำหรับฐานสิบ (Decimal System) เราจะแยกตัวเลขตามหลักหน่วย หลักสิบ หรือหลักร้อย ซึ่งค่าของแต่ละหลักจะอยู่ในช่วง 0 ถึง 9 ทำให้ต้องการ 10 แถวใน Radix[] เพื่อรองรับค่าของหลักเหล่านี้
- #define MaxCol 20 : กำหนดจำนวนคอลัมน์ (Column) ใน array Radix ซึ่งในที่นี้กำหนดเป็น 20 หมายความว่าแต่ละแถวใน Radix[] สามารถเก็บตัวเลขได้สูงสุด 20 ตัวเลขในแต่ละหลัก

## 3. การประกาศตัวแปร

```
int Data[MaxData];

int Radix[MaxRow][MaxCol]; //Radix เป็นถึงเก็บชั่วคราว ขนาดคือ [0..MaxRow, 0..MaxCol]

int N, N1;
```

ในส่วนการประกาศตัวแปร มีรายละเอียดดังนี้

- int Data[MaxData] : ประกาศ array ชื่อ Data[] ที่มีขนาดเท่ากับ MaxData (ซึ่งก็คือ 100) ใช้สำหรับเก็บตัวเลขที่สุ่มขึ้นมาเพื่อใช้ในกระบวนการจัดเรียงข้อมูลด้วย Radix Sort โดยแต่ละช่องใน array นี้จะเก็บตัวเลขที่มี 3 หลัก (ในช่วง 100 ถึง 999)
- int Radix[MaxRow][MaxCol] : ประกาศ array สองมิติชื่อ Radix[][] ขนาด MaxRow x MaxCol (ซึ่งก็คือ 10 x 20) ทำหน้าที่เป็นถึงเก็บชั่วคราวสำหรับข้อมูลในแต่ละหลัก (Digit) ของตัวเลข เมื่อโปรแกรมจัดเรียงตัวเลขตามหลักหน่วย หลักสิบ หรือหลักร้อย ข้อมูลจะถูกแยกเก็บในแถวต่าง ๆ ของ

Radix ตามค่าของหลักนั้น ๆ โดย แถว (Row) แทนค่าของหลักหน่วย (0-9) และคอลัมน์ (Column) แทนตำแหน่งของตัวเลขในแต่ละแถว

- int N, N1 : ตัวแปร N ใช้สำหรับเก็บจำนวนข้อมูลที่เราต้องการจัดเรียง (ในกรณีนี้ N = 16 คือจำนวนข้อมูล 16 ตัวที่สุ่มขึ้นมา) และ ตัวแปร N1 ใช้เก็บค่า N ไว้สำหรับการแสดงผลข้อมูลหลังจากการจัดเรียงเสร็จสิ้น เนื่องจากในบางครั้งค่าของ N อาจถูกเปลี่ยนแปลงในกระบวนการจัดเรียง

#### 4. ฟังก์ชัน ClearStackPT

```
void ClearStackPT() //ล้างทุกบล็อกเป็น 0 และใช้ Radix[0] สำหรับ SP
{
    int i;

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

        Radix[i][0] = NULL; //ตำแหน่งนี้เก็บ SP
}
```

ฟังก์ชัน ClearStackPT ทำหน้าที่ "ล้างข้อมูล" ในสแต็กที่เก็บใน array Radix[][] ซึ่งเป็นอาร์เรย์สองมิติที่ใช้เก็บข้อมูลชั่วคราวขณะที่ข้อมูลกำลังถูกจัดเรียงตามแต่ละหลักของตัวเลข โดยทุกครั้งที่เสร็จสิ้นการจัดเรียงในแต่ละหลัก ข้อมูลใน Radix จำเป็นต้องถูกเคลียร์เพื่อให้พร้อมสำหรับจัดเรียงในหลักถัดไป โดยหลักการทำงานฟังก์ชันจะวนลูปผ่านทุกแถว (i ตั้งแต่ 0 ถึง MaxRow ซึ่งเป็น 9) ของ Radix ซึ่งทุกแถวจะมีคอลัมน์แรก (Radix[i][0]) เก็บค่าตัวชี้ตำแหน่งของข้อมูลในแถว (เหมือนเป็น Stack Pointer - SP) โดยค่าที่อยู่ในตำแหน่ง Radix[i][0] จะถูกตั้งค่าให้เป็น NULL เพื่อบอกว่าข้อมูลในแต่ละแถวได้ถูกเคลียร์แล้ว

#### 5. ฟังก์ชัน PrepareRawData

```
void PrepareRawData(int N2)
{
    int i;

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

    for (i = 0; i < N2; i++)
```

## 5. ฟังก์ชัน PrepareRawData (ต่อ)

```
Data[i] = (rand() % 899) + 100; //สุ่มตัวเลขที่ต่างกันในช่วง 100..999
}
```

ฟังก์ชัน PrepareRawData ใช้สำหรับสร้างข้อมูลดิบ (Raw Data) ที่เป็นข้อมูลสุ่มในช่วง 100 ถึง 999 แล้วเก็บไว้ใน array Data[] เพื่อใช้ในการทดสอบการทำงานของการทำงานของเครื่อง โดยหลักการทำงานของเครื่องจะใช้ฟังก์ชัน srand(time(NULL)) เพื่อกำหนดค่า Seed สำหรับการสุ่มตัวเลข โดย time(NULL) จะคืนค่าที่แตกต่างกันทุกครั้ง (เวลาปัจจุบันในหน่วยวินาที) เพื่อให้ได้ค่าตัวเลขสุ่มที่ไม่ซ้ำกันในแต่ละครั้งที่รันโปรแกรม จากนั้นจะวนลูปตั้งแต่  $i = 0$  จนถึง  $N2-1$  เพื่อสุ่มตัวเลขโดยใช้ฟังก์ชัน rand() ซึ่งคืนค่าตัวเลขสุ่มในช่วง 0 ถึง 899 จากนั้นเพิ่มค่า 100 เพื่อให้ผลลัพธ์อยู่ในช่วง 100 ถึง 999 แล้วเก็บผลลัพธ์ไว้ใน array Data[i]

## 6. ฟังก์ชัน DispData

```
void DispData(int N2) //แสดงข้อมูลในอาร์เรย์
{
    int i;

    for(i = 0; i < N2; i++)

        printf("%3d ", Data[i]);

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

ฟังก์ชัน DispData ทำหน้าที่แสดงข้อมูลใน array Data[] ออกทางหน้าจอ เพื่อให้เห็นว่าข้อมูลในอาร์เรย์มีค่าตัวเลขอะไรอยู่บ้าง ทั้งก่อนและหลังการจัดเรียง โดยหลักการทำงานของฟังก์ชันนี้จะวนลูปตั้งแต่  $i = 0$  ถึง  $N2-1$  เพื่อแสดงข้อมูลแต่ละตัวที่เก็บอยู่ใน Data[] จากนั้นใช้ printf แสดงตัวเลขที่อยู่ใน array Data[i] โดยจัดรูปแบบให้แต่ละตัวเลขแสดงเป็นตัวเลข 3 หลัก (%3d) เพื่อให้ดูเป็นระเบียบ

## 7. ฟังก์ชัน Push

```
void Push(int Rad, int Dat) //ใส่ข้อมูลลงใน Stack แบบขนาน โดยเก็บ SP ที่ (Rad, 0)
{
    int SP;

    SP = Radix[Rad][0] + 1; //ข้าม SP

    Radix[Rad][0] = SP; //เก็บ SP ใหม่

    Radix[Rad][SP] = Dat; //ดันข้อมูลลงไป Radix นั้น
}
```

ฟังก์ชัน Push ใช้ในการเก็บข้อมูล Dat ลงใน Radix ซึ่งเปรียบเสมือนเป็น Stack (สแต็ก) ของตัวเลขที่ถูกจัดเรียงตามหลัก โดยใช้หลักของตัวเลขเป็นตัวบอกว่าควรเก็บข้อมูลใน Radix ไດ โดยหลักการทำงาน ฟังก์ชันจะดึงค่าตัวชี้ตำแหน่ง (SP) ในแต่ละ Radix[Rad][0] ซึ่งบ่งบอกถึงตำแหน่งล่าสุดในแถวที่ยังว่างอยู่ จากนั้นจะเพิ่มค่า SP ขึ้นหนึ่งตำแหน่ง (SP+1) โดยค่าของ Radix[Rad][SP] จะถูกเก็บข้อมูลตัวเลข (Dat) ที่ได้รับเข้ามา แล้วอัปเดต Radix[Rad][0] เพื่อเก็บตำแหน่ง SP ใหม่ จากนั้นตัวเลขจะถูกเก็บในแถวที่สัมพันธ์กับค่าของหลักนั้น ๆ เช่น หากหลักหน่วยของตัวเลขคือ 5 ข้อมูลจะถูกเก็บใน Radix[5]

## 8. ฟังก์ชัน ReadStack

```
void ReadStack() //อ่านข้อมูลจากแต่ละ Stack และถ่ายโอนไปยัง Data[]
{
    int i, j, k, SP;

    k = 0; //เริ่มที่ Data[] ตำแหน่งแรกคือ 0

    for(i = 0; i <= MaxRow; i++)
    {
        SP = Radix[i][0]; //ค่าจำนวนสูงสุดของแต่ละ Radix เก็บที่คอลัมน์ 0

        for(j = 1; j <= SP; j++)
```

## 8. ฟังก์ชัน ReadStack (ต่อ)

```

{
    Data[k] = Radix[i][j]; //ถ่ายโอนข้อมูลจากทุก Stack ไปยัง Data[]

    k++;

}

} //จบ for

} //จบฟังก์ชัน

```

ฟังก์ชัน ReadStack ทำหน้าที่นำข้อมูลจาก Radix แต่ละแถว (ซึ่งเก็บข้อมูลที่จัดเรียงตามหลักหน่วยหรือหลักอื่น ๆ) กลับคืนมาสู่ array Data[] เพื่อใช้ในการจัดเรียงต่อในหลักถัดไป โดยหลักการทำงาน ฟังก์ชันจะวนลูปผ่านทุกแถวใน Radix ( $i = 0$  ถึง MaxRow) โดยแต่ละแถวจะมีค่าตัวชี้ตำแหน่ง (SP) ที่บอกจำนวนข้อมูลที่ถูกเก็บไว้ในแถว โดยภายในแต่ละแถวจะมีการวนลูปซ้อน ( $j = 1$  ถึง SP) เพื่อดึงข้อมูลจาก Radix[i][j] แต่ละตำแหน่งกลับมาใส่ใน Data[] โดยเริ่มต้นจากตำแหน่งแรกสุดใน array Data[] (เก็บในตัวแปร k ซึ่งจะถูกเพิ่มขึ้นเรื่อย ๆ)

## 9. ฟังก์ชัน RadixSort

```

void RadixSort(int N2)
{
    int Digit, i, RadixNo;

    char Txt[2];

    for(Digit = 2; Digit >= 0; Digit--) //นับถอยหลัง
    {
        printf("[Digit : %d]==>\n", 3 - Digit);

        for(i = 0; i < N2; i++) //ตัวนับ Data[]
        {

```

## 9. ฟังก์ชัน RadixSort (ต่อ)

```

        itoa(Data[i], Txt, 10); //แปลง Integer เป็น Text [itoa(อินพุต, เอาท์พุต, ฐาน)]

        RadixNo = Txt[Digit] - 48; //แปลง Text ของหลักนั้นให้เป็นตัวเลข

        Push(RadixNo, Data[i]); //ดันข้อมูลไปยัง Stack ใน Radix นั้น

    } //จบ for

    ReadStack();

    DispData(N2);

    ClearStackPT();

} //จบ for

} //จบฟังก์ชัน

```

ฟังก์ชัน RadixSort เป็นฟังก์ชันหลักของโปรแกรมที่ทำหน้าที่จัดเรียงข้อมูลใน array Data[] ตามเทคนิค Radix Sort โดยจะเริ่มต้นจัดเรียงจากหลักหน่วย แล้วไปยังหลักสิบและหลักร้อยตามลำดับ โดยหลักการทำงานเริ่มต้นจากวนลูปหลักการจัดเรียง (Digit = 2 ถึง 0) ซึ่งแสดงถึงการจัดเรียงจากหลักร้อย (หลักที่ 2) ไปจนถึงหลักหน่วย (หลักที่ 0) ในแต่ละรอบ จะมีการแยกตัวเลขตามค่าของหลักปัจจุบัน ต่อมาจะใช้ฟังก์ชัน itoa() เพื่อแปลงค่าตัวเลขใน array Data[] เป็นสตริง เช่น ตัวเลข 352 จะแปลงเป็นสตริง "352" เพื่อที่จะสามารถเข้าถึงหลักต่าง ๆ ได้อย่างง่ายดาย จากนั้นดึงตัวเลขในหลักที่ต้องการ (หลักที่ Digit) ด้วยการลบค่า ASCII ของตัวเลข (Txt[Digit] - 48) เพื่อเปลี่ยนค่าจากสตริงเป็นตัวเลข ในส่วนของข้อมูลจาก Data[] แต่ละตัว จะถูกดันเข้าสู่ Radix ที่ตรงกับค่าของหลักปัจจุบัน โดยใช้ฟังก์ชัน Push() เช่น หากค่าหลักหน่วยคือ 5 ตัวเลข นั้นจะถูกเก็บใน Radix[5] หลังจากจัดเก็บข้อมูลใน Radix เสร็จ ฟังก์ชัน ReadStack() จะถูกเรียกเพื่อดึงข้อมูลจาก Radix กลับไปใส่ใน Data[] โดยเรียงลำดับจาก Radix แฉวที่น้อยไปมาก หลังจากจัดเรียงข้อมูลในแต่ละหลัก (หน่วย, สิบ, ร้อย) เสร็จสิ้น ข้อมูลใน Radix ต้องถูกเคลียร์เพื่อเตรียมพร้อมสำหรับการจัดเรียงในหลักถัดไป ฟังก์ชัน ClearStackPT() จะถูกเรียกเพื่อรีเซ็ตค่า Stack Pointer (SP) ในแต่ละแฉวของ Radix กลับเป็น NULL เมื่อเสร็จสิ้นการจัดเรียงสำหรับหลักหนึ่ง (เช่น หลักหน่วย) ฟังก์ชัน RadixSort() จะวนไปจัดเรียงตามหลักถัดไป (หลักสิบ, หลักร้อย) โดยทำซ้ำขั้นตอนเดิมจนจัดเรียงครบทุกหลักที่ต้องการ



## 10. ฟังก์ชัน main

```

int main()
{
    printf("ASCENDING RADIX SORT\n");

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

    N = 16; //เปลี่ยนจำนวน N ที่นี่

    N1 = N; //เก็บค่า N ไปยัง N1 เพราะ N จะเป็น 0 เมื่อเสร็จสิ้น RadixSort()

    PrepareRawData(N);

    printf("Raw Data...\n");

    DispData(N);

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

    printf("Processing Data...\n");

    RadixSort(N);

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

    printf("Sorted Data : \n");

    DispData(N1); //ข้อมูลที่จัดเรียงแล้ว

    getch();

    return(0);

} //จบ Main

```

ฟังก์ชันหลัก (main) เป็นส่วนที่เรียกใช้การทำงานของโปรแกรม โดยจะกำหนดจำนวนข้อมูลที่ต้องการจัดเรียง สุ่มข้อมูลดิบ แสดงผลลัพธ์ก่อนและหลังการจัดเรียง รวมถึงการแสดงรายละเอียดของการจัดเรียงในแต่ละขั้นตอน โดยหลักการทำงานเริ่มต้นด้วย จำนวนข้อมูลที่ต้องการจัดเรียงถูกตั้งค่าไว้ที่ N = 16 ซึ่งหมายความว่าโปรแกรมจะสุ่มตัวเลข 16 ตัวในช่วง 100-999 เพื่อจัดเรียง จากนั้นฟังก์ชัน

PrepareRawData(N) จะถูกเรียกเพื่อสุ่มตัวเลขดิบ 16 ตัวเก็บไว้ใน Data[] เมื่อเรียกใช้งานฟังก์ชัน PrepareRawData(N) เสร็จจะใช้งานฟังก์ชัน DispData(N) จะถูกเรียกเพื่อแสดงข้อมูลดิบที่สุ่มขึ้นมาก่อนการจัดเรียง เพื่อให้เห็นสภาพก่อนการจัดเรียง จากนั้นจะใช้งานฟังก์ชัน RadixSort(N) จะถูกเรียกเพื่อจัดเรียงข้อมูลใน Data[] โดยจะจัดเรียงข้อมูลจากหลักร้อยไปยังหลักหน่วย และแสดงขั้นตอนการจัดเรียงในแต่ละรอบ เมื่อเสร็จสิ้นการจัดเรียง ฟังก์ชัน DispData(N1) จะถูกเรียกเพื่อแสดงผลลัพธ์สุดท้ายที่ผ่านการจัดเรียงเรียบร้อยแล้ว

## ผลลัพธ์การใช้งานโปรแกรม RADIX SORT

โปรแกรม RADIX SORT ใช้เทคนิค Radix Sort มีวัตถุประสงค์เพื่อจัดเรียงชุดข้อมูลจำนวนหนึ่งจากน้อยไปหามาก โดยที่ผู้ใช้ไม่จำเป็นต้องทำการเปรียบเทียบระหว่างตัวเลขโดยตรง แต่จะทำการจัดเรียงตามหลักต่าง ๆ ของตัวเลข ซึ่งในกรณีนี้มีการกำหนดให้จัดเรียงตัวเลขที่สุ่มขึ้นมาในช่วง 100 ถึง 999 โปรแกรมนี้ช่วยให้ผู้ใช้เห็นถึงขั้นตอนการทำงานและผลลัพธ์ที่เกิดขึ้นหลังจากการจัดเรียงอย่างชัดเจน

### 1. การเริ่มต้นของโปรแกรม

เมื่อเริ่มต้นรันโปรแกรม จะมีการพิมพ์ข้อความ “ASCENDING RADIX SORT” เพื่อแจ้งให้ผู้ใช้ทราบว่าโปรแกรมกำลังดำเนินการเรียงลำดับข้อมูลแบบ Heap Sort โดยจัดเรียงจากน้อยไปมาก (Ascending)

```
E:\ENGCE124\Coding 22 RADIX SORT
ASCENDING RADIX SORT
=====
```

### 2. การเตรียมข้อมูลดิบสำหรับแสดงข้อมูล

เมื่อโปรแกรมเริ่มทำงาน จะมีการเรียกใช้ฟังก์ชัน PrepareRawData(N) เพื่อสุ่มตัวเลขจำนวน 16 ตัวในช่วง 100 ถึง 999 โดยใช้ rand() ในการสุ่ม ตัวเลขเหล่านี้จะถูกเก็บไว้ใน array Data[] ซึ่งมีขนาดสูงสุด 100 ตัวเลข หลังจากที่เราสุ่มตัวเลขเสร็จสิ้น จะมีการแสดงผลข้อมูลดิบที่สุ่มขึ้นมา โดยใช้ฟังก์ชัน DispData(N) ข้อมูลดิบจะแสดงในรูปแบบของตัวเลขที่ถูกจัดเรียงเป็นแถว ซึ่งช่วยให้ผู้ใช้เห็นสภาพข้อมูลก่อนการจัดเรียง

```
Raw Data...
674 810 410 677 296 216 250 483 285 864 217 185 579 422 172 499
=====
```

### 3. การจัดเรียงข้อมูล

เมื่อข้อมูลดิบแสดงผลเสร็จแล้ว โปรแกรมจะเริ่มกระบวนการจัดเรียงโดยเรียกใช้ฟังก์ชัน RadixSort(N) โดยโปรแกรมจะทำการจัดเรียงข้อมูลตามหลักต่าง ๆ โดยเริ่มจากหลักหน่วยไปยังหลักสิบและหลักร้อย ซึ่งในแต่ละรอบของการจัดเรียง โปรแกรมจะแสดงผลลัพธ์ในแต่ละขั้นตอน

```
Processing Data...
[Digit : 1]==>
810 410 250 422 172 483 674 864 285 185 296 216 677 217 579 499
[Digit : 2]==>
810 410 216 217 422 250 864 172 674 677 579 483 285 185 296 499
[Digit : 3]==>
172 185 216 217 250 285 296 410 422 483 499 579 674 677 810 864
-----Finished
```

#### 4. การแสดงผลหลังการจัดเรียง

เมื่อการจัดเรียงเสร็จสิ้น โปรแกรมจะทำการแสดงผลข้อมูลที่ถูกจัดเรียงแล้วโดยใช้ฟังก์ชัน `DispData(N1)` โดยผลลัพธ์จะแสดงข้อมูลที่ถูกจัดเรียงจากน้อยไปหามากในรูปแบบตัวเลข

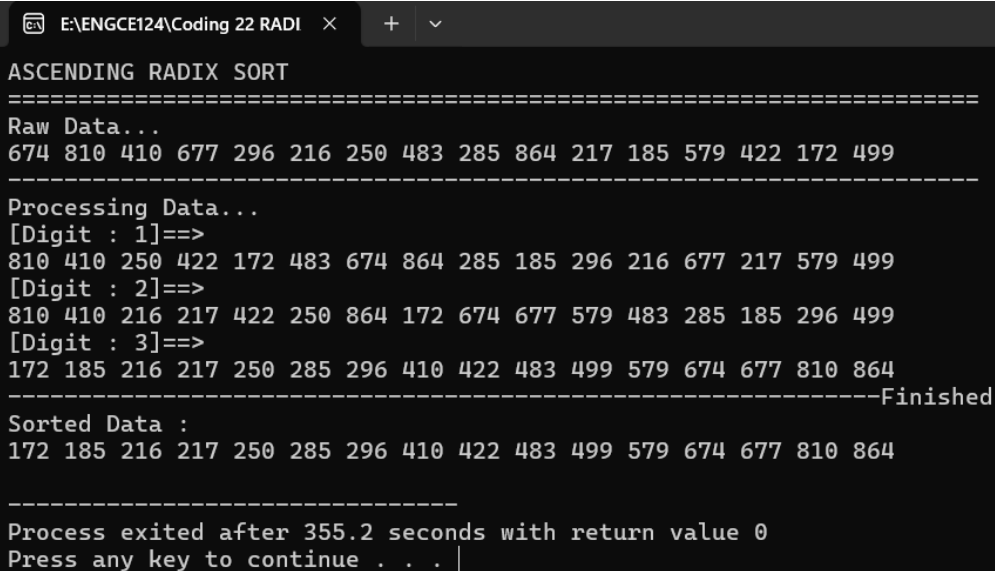
```
Sorted Data :
172 185 216 217 250 285 296 410 422 483 499 579 674 677 810 864
|
```

#### 5. การจบการทำงานของโปรแกรม

โปรแกรมจะหยุดการทำงานชั่วคราวและรอการกดปุ่มใดๆ จากผู้ใช้งานก่อนที่จะปิดโปรแกรม (ผ่านฟังก์ชัน `getch()`) ซึ่งเป็นการจบการทำงานของโปรแกรม

```
-----
Process exited after 355.2 seconds with return value 0
Press any key to continue . . . |
```

ภาพรวมผลลัพธ์ของโปรแกรม RADIX SORT



```
E:\ENGCE124\Coding 22 RADIX  ×  +  ▾

ASCENDING RADIX SORT
=====
Raw Data...
674 810 410 677 296 216 250 483 285 864 217 185 579 422 172 499
-----
Processing Data...
[Digit : 1]==>
810 410 250 422 172 483 674 864 285 185 296 216 677 217 579 499
[Digit : 2]==>
810 410 216 217 422 250 864 172 674 677 579 483 285 185 296 499
[Digit : 3]==>
172 185 216 217 250 285 296 410 422 483 499 579 674 677 810 864
-----Finished
Sorted Data :
172 185 216 217 250 285 296 410 422 483 499 579 674 677 810 864

-----
Process exited after 355.2 seconds with return value 0
Press any key to continue . . . |
```

### บรรณานุกรม

ChatGPT. ( - ). Understanding Radix Sort: Process and Results. สืบค้น 22 กันยายน 2567,  
จาก <https://chatgpt.com/>