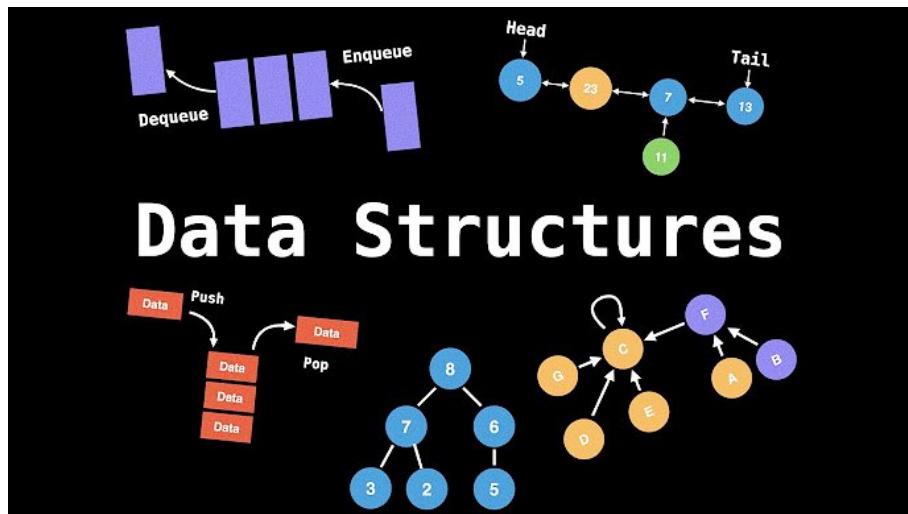


# Linear Data Structure Report

CPE112



Thinnaphat Kanchina 66070501013

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

# Week 1

## Lab 1 Warm Up!

### Problem #1 | To be set

ปัญหาคือแปลง array เป็น set โดยในโค้ดจะมีฟังก์ชันหลักคือฟังก์ชันแปลงอาร์เรย์เป็นเซ็ต (arr2set) ฟังก์ชันนี้รับอาร์เรย์ arr และขนาดของอาร์เรย์ size เป็นอินพุตและส่งคืนอาร์เรย์ใหม่ที่มีเฉพาะค่าที่ไม่ซ้ำกันใน arr

```
int *arr2set(int *arr, int *size)
{
    int i, j, k;
    for (i = 0; i < (*size); i++)
    {
        for (j = i + 1; j < (*size); j++)
        {
            if (arr[i] == arr[j])
            {
                for (k = j; k < (*size); k++)
                {
                    arr[k] = arr[k + 1];
                }
                (*size)--;
                j--;
            }
        }
    }
    int *set = (int *)malloc((*size) * sizeof(int));
    for (i = 0; i < *size; i++)
    {
        set[i] = arr[i];
    }
    return set;
}
```

วิธีการทำงาน:

- 1.วนซ้ำแต่ละตัวใน arr เพรียบเทียบกับตัวถัดไป
- 2.ถ้าเจอค่าที่ซ้ำกัน จะลบตัวถัดไปออกจาก arr และลดขนาดของอาร์เรย์ลง
- 3.ทำซ้ำจนกว่าจะไม่มีค่าซ้ำกันใน arr
- 4.สร้างอาร์เรย์ใหม่ set เก็บค่าที่ไม่ซ้ำกันจาก arr
- 5.ส่งคืน set

### Problem #2 | The foundation of set

โจทย์ต้องการให้ทำการ Union, Intersection, Difference, และ Complement ระหว่างเซ็ตที่กำหนดใน universe ที่ระบุ (จำนวนเต็ม m ถึง n) โดยต้องแปลงอาร์เรย์ให้เป็นเซ็ตและตรวจสอบว่าผลลัพธ์เป็นเซ็ตว่างหรือไม่ (ถ้าเป็นเซ็ตว่างให้พิมพ์ “empty” ) ในโค้ดนี้จะมีฟังก์ชันหลักอยู่ 4 ฟังก์ชันตาม operation ของ set

```
int *get_union(int *set_a, int *set_b, int *size_a, int *size_b, int union_size)
{
    int i, j, k;
    int *union_set = (int *)malloc((union_size) * sizeof(int));
    for (i = 0; i < *size_a; i++)
    {
        union_set[i] = set_a[i];
    }
    for (j = 0; j < *size_b; j++)
    {
        union_set[i + j] = set_b[j];
    }
    union_set = arr2set(union_set, union_size);
    union_set = sort_set(union_set, union_size);
    return union_set;
}
```

get\_union Function:

การทำงาน: รับเซ็ต A, เซ็ต B, ขนาดของ A, ขนาดของ B, และขนาดของเซตสหวัต (union set) เป็นพารามิเตอร์ จากนั้นสร้างเซตสหวัต (union set) โดยรวมทุกสมาชิกจาก A และ B, จากนั้นใช้ arr2set เพื่อลบข้อมูลที่ซ้ำกัน และ sort\_set เพื่อเรียงลำดับสมาชิก

เอกสาร์พต: Union set

```

int *get_intersection(int *set_a, int *set_b, int *size_a, int *size_b, int *intersection_size)
{
    int i, j, k;
    int *intersection_set = (int *)malloc((*intersection_size) * sizeof(int));
    int num_intersection = 0;
    for (i = 0; i < *size_a; i++)
    {
        for (j = 0; j < *size_b; j++)
        {
            if (set_a[i] == set_b[j])
            {
                intersection_set[i] = set_a[i];
                num_intersection++;
                (*intersection_size)--;
            }
        }
    }
    *intersection_size -= ((*size_b) - num_intersection) + ((*size_a) - num_intersection);
    intersection_set = arr2set(intersection_set, intersection_size);
    intersection_set = sort_set(intersection_set, intersection_size);

    return intersection_set;
}

```

### get\_intersection Function:

การทำงาน: รับเซต A, เซต B, ขนาดของ A, ขนาดของ B, และขนาดของเซตที่เก็บส่วนกลับกัน (intersection set) เป็นพารามิเตอร์ จากนั้นหาสมาชิกที่ซ้ำกันระหว่าง A และ B และลบข้อมูลที่ไม่ซ้ำกัน, จากนั้นใช้ remove\_duplicates เพื่อลบข้อมูลที่ซ้ำกัน และ sort\_set เพื่อเรียงลำดับสมาชิก  
ส่งคืน: เซตที่มีสมาชิกที่เป็นส่วนกลับกัน (intersection set)

```

int *get_difference(int *set_a, int *set_b, int *size_a, int *size_b, int *difference_size)
{
    int i, j, k;
    int *difference_set = (int *)malloc((*difference_size) * sizeof(int));
    for (i = 0; i < *size_a; i++)
    {
        difference_set[i] = set_a[i];
    }
    for (i = 0; i < *size_a; i++)
    {
        for (j = 0; j < *size_b; j++)
        {
            if (difference_set[i] == set_b[j])
            {
                for (k = i; k < *size_a; k++)
                {
                    difference_set[k] = difference_set[k + 1];
                }
                (*difference_size)--;
            }
        }
    }
    difference_set = arr2set(difference_set, difference_size);
    difference_set = sort_set(difference_set, difference_size);

    return difference_set;
}

```

### get\_difference Function:

การทำงาน: รับเซต A, เซต B, ขนาดของ A, ขนาดของ B, และขนาดของเซตที่เก็บความแตกต่าง (difference set) เป็นพารามิเตอร์ จากนั้นหาสมาชิกที่มีใน A แต่ไม่มีใน B และลบข้อมูลที่ซ้ำกัน, จากนั้นใช้ remove\_duplicates เพื่อลบข้อมูลที่ซ้ำกัน และ sort\_set เพื่อเรียงลำดับสมาชิก  
ส่งคืน: เซตที่มีสมาชิกที่เป็นความแตกต่าง (difference set)

```

int *get_complement(int *set, int *size, int *uni_set, int *uni_size, int *complement_size)
{
    int i, j, k;
    int *complement_set = (int *)malloc((*uni_size) * sizeof(int));
    for (i = 0; i < *uni_size; i++)
    {
        complement_set[i] = uni_set[i];
    }

    for (i = 0; i < *size; i++)
    {
        for (j = 0; j < *uni_size; j++)
        {
            if (set[i] == complement_set[j])
            {
                for (k = j; k < *uni_size; k++)
                {
                    complement_set[k] = complement_set[k + 1];
                }
            }
        }
    }

    complement_set = arr2set(complement_set, complement_size);
    complement_set = sort_set(complement_set, complement_size);

    return complement_set;
}

```

### get\_complement Function:

การทำงาน: รับเซตที่ต้องการหาค่า complement, ขนาดของเซต, universal set, ขนาดของ universal set, และขนาดของเซตที่เก็บ complement เป็นพารามิเตอร์ จากนั้นหาสมาชิกที่ไม่มีในเซตที่ต้องการหาค่า complement และลบข้อมูลที่ซ้ำกัน, จากนั้นใช้ remove\_duplicates เพื่อลบข้อมูลที่ซ้ำกัน และ sort\_set เพื่อเรียงลำดับสมาชิก  
ส่งคืน: เซตที่มีสมาชิกเป็นค่า complement

# Week 2

## Lab 2 Array

### Problem #1 | No Bracket

โจทย์นี้ต้องเขียนโปรแกรมเพื่อพิมพ์อาร์เรย์โดยไม่ใช้ 'square brackets' และตัวเลือกการพิมพ์ตามดังนี้: 0 พิมพ์ค่าที่มีดัชนีเลขคู่ 1 พิมพ์ค่าที่มีดัชนีเลขคี่ ถ้าไม่มีค่าในดัชนีที่ต้องการพิมพ์ ให้พิมพ์ 'none'

```
int main()
{
    int n;
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++)
    {
        scanf("%d", arr + i);
    }
    int mode;
    scanf("%d", &mode);
    if (mode == 0)
    {
        for (int i = 0; i < n; i += 2)
        {
            printf("%d ", *(arr + i));
        }
    }
    else if (mode == 1)
    {
        if (n > 1)
        {
            for (int i = 1; i < n; i += 2)
            {
                printf("%d ", *(arr + i));
            }
        }
        else
        {
            printf("none");
        }
    }
    free(arr);
    return 0;
}
```

#### โดยเทคนิคที่ใช้คือ

- ใช้ pointer เพื่อรับและเข้าถึงข้อมูลในอาร์เรย์
- ในลูป for, ให้ดัชนีเริ่มต้นตามโหมดที่ระบุ (0 หรือ 1) และเพิ่มขึ้นทีละ 2 เพื่อพิมพ์ค่าที่มีดัชนีเลขคู่หรือคี่
- ถ้าไม่มีข้อมูลที่จะพิมพ์ ( $n \leq 1$  หรือ  $mode \geq n$ ), ให้พิมพ์ "none"
- ใช้ free เพื่อคืนทรัพยากรที่ถูกจองสำหรับอาร์เรย์

### Problem #2 | No Bracket No Printf

โปรแกรมต้องพิมพ์อาร์เรย์โดยไม่ใช้ 'square brackets', ไม่ใช้ printf() ใน main function, และประกาศอาร์เรย์ใน main function เท่านั้น. หลังจากนั้นหาค่าสูงสุดและต่ำสุดในอาร์เรย์พร้อม index ที่เริ่มต้นที่ 0, ถ้ามีการซ้ำให้ให้พิมพ์ตัวนีที่มีค่าน้อยที่สุด โดยเทคนิคที่ใช้หลักๆ จะมีดังนี้

```
int get_max(int *arr, int size)
{
    int max = *arr;
    for (int i = 1; i < size; i++)
    {
        if (*(arr + i) > max)
        {
            max = *(arr + i);
        }
    }
    return max;
}
int get_min(int *arr, int size)
{
    int min = *arr;
    for (int i = 1; i < size; i++)
    {
        if (*(arr + i) < min)
        {
            min = *(arr + i);
        }
    }
    return min;
}
void show_max_min_and_idx_of_array(int *arr, int size)
{
    int max_num = get_max(arr, size);
    int max_idx = get_idx(arr, size, max_num);
    int min_num = get_min(arr, size);
    int min_idx = get_idx(arr, size, min_num);
    printf("%d %d\n%d %d", max_num, max_idx, min_num, min_idx);
}
```

#### การแสดงผลโดยไม่ใช้ printf:

ใช้การ printf ในฟังก์ชัน show\_max\_min\_and\_idx\_of\_array แทนเพื่อแสดงผล

#### การเข้าถึงข้อมูลโดยไม่ใช้ []:

ในฟังก์ชัน get\_max, get\_min, และ get\_idx, ใช้ pointer แทนการใช้ [] เพื่อเข้าถึงข้อมูลในอาร์เรย์

#### แสดงผลโดยไม่ใช้ 'square brackets':

การเรียกใช้ด้วย memory address ทำให้ไม่ต้องใช้ 'square brackets' เพื่อพิมพ์ข้อมูล

## Problem #3 | Sum of Diagonal Matrix

คำนวณผลรวมของเลขบนเส้นทแยงหลัก (primary diagonal) และเส้นทแยงรอง (secondary diagonal) ของเมทริกซ์ โปรแกรมรับขนาดของเมทริกซ์ (จำนวนแถวและคอลัมน์) และข้อมูลภายในเมทริกซ์เป็นอินพุต กรณีที่จำนวนแถวไม่เท่ากับจำนวนคอลัมน์ ให้โปรแกรมแสดงข้อความ "ERROR" และหยุดการทำงาน

```
int get_primary_diagonal(int **arr, int row, int col){
    int sum = 0;
    for (int i = 0; i < row; i += 1){
        sum += arr[i][i];
    }
    return sum;
}
int get_secondary_diagonal(int **arr, int row, int col){
    int sum = 0;
    for (int i = 0; i < row; i += 1){
        sum += arr[i][col - i - 1];
    }
    return sum;
}
int main(){
    int row, col;
    scanf("%d %d", &row, &col);
    if (row != col){
        printf("ERROR\n");
        return 0;
    }
    int **matrix = (int **)malloc(row * sizeof(int *));
    for (int i = 0; i < row; i += 1){
        matrix[i] = (int *)malloc(col * sizeof(int));
        for (int j = 0; j < col; j += 1){
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("Primary: %d\n", get_primary_diagonal(matrix, row, col));
    printf("Secondary: %d\n", get_secondary_diagonal(matrix, row, col));
    free(matrix);
    return 0;
}
```

รายละเอียดโค้ด:

- **get\_primary\_diagonal:** พังก์ชันนี้คำนวณผลรวมของเลขบนเส้นทแยงหลัก โดยวนลูป *i* จาก 0 ถึง *row-1* และรวมค่าที่ตำแหน่ง *arr[i][i]*
- **get\_secondary\_diagonal:** พังก์ชันนี้คำนวณผลรวมของเลขบนเส้นทแยงรอง โดยใช้สูตร *arr[i][col - i - 1]* โดย *i* แทนแถวและ *col* แทนจำนวนคอลัมน์ทั้งหมด
- **main:** พังก์ชันหลักรับขนาดของเมทริกซ์ (*row* และ *col*) และตรวจสอบขนาดเท่ากันหรือไม่ ถ้าไม่เท่า พิมพ์ "ERROR" ถ้าเท่า จัดสรรพื้นที่สำหรับเมทริกซ์แบบสองมิติ และในลูปรับค่าในเมทริกซ์ สุดท้าย เรียกใช้ *get\_primary\_diagonal* และ *get\_secondary\_diagonal* เพื่อคำนวณผลรวมบนเส้นทแยงหลักและรองตามลำดับ

## Problem #4 | Dictionary

จำลองการทำงานของพจนานุกรม พจนานุกรมนี้ใช้โครงสร้างข้อมูลแบบ key-value pairs key เป็นเหมือนชื่อเรียก ข้อมูล (string) value เป็นข้อมูลที่ต้องการเก็บ (string) สามารถเพิ่มข้อมูล (key-value) ใหม่และสามารถแก้ไขข้อมูลที่มีอยู่แล้ว โดยระบุ key

```
struct dict
{
    char value[100];
    char key[100];
};
void editdict(struct dict *dic, int size, char *key, char *value){
    int change_count = 0;
    for (int i = 0; i < size; i++){
        if (strcmp(dic[i].key, key) == 0)
        {
            strcpy(dic[i].value, value);
            change_count++;
        }
    }
    if (change_count == 0)
    {
        printf("No change\n");
    }
}
void printdict(struct dict *dic, int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%s : %s\n", dic[i].key, dic[i].value);
    }
}
```

### การทำงาน

- รับขนาดพจนานุกรม (size) จำนวนคู่ของ key-value ( $1 \leq \text{size} \leq 10$ )
- รับข้อมูล key-value แต่ละคู่แยกด้วยช่องว่าง (whitespace)
- รับข้อมูล key ที่ต้องการแก้ไขและข้อมูล value ใหม่
- แสดงผลลัพธ์
- กรณีแก้ไขข้อมูลสำเร็จ: แสดงข้อมูลทั้งหมดในพจนานุกรมหลังแก้ไข (key:value)
- กรณีไม่พบ key ที่ต้องการแก้ไข: แสดงข้อความ "No change" ตามด้วยข้อมูลทั้งหมดในพจนานุกรมเดิม (key:value)

## Assignment 2 Array

### Problem #1 | Grading

เก็บข้อมูลของนักเรียน ประกอบด้วย ชื่อ และคะแนน  
จากนั้นคำนวณข้อมูล ดังนี้

- คะแนนเฉลี่ย ของนักเรียนทั้งหมด (หาโดยใช้สูตร  $(\text{ผลรวมคะแนน}) / (\text{จำนวนนักเรียน})$ )
- ส่วนเบี่ยงมาตรฐาน ของคะแนน (หาโดยใช้สูตร  $\sqrt{(\text{ผลรวม} (\text{คะแนนแต่ละคน} - \text{คะแนนเฉลี่ย})^2) / (\text{จำนวนนักเรียน})}$ )
- ชื่อนักเรียน ที่ได้คะแนน สูงสุด และ ต่ำสุด

```
struct Student {
    char name[10];
    float score;
};

int main() {
    int size;
    scanf("%d", &size);

    struct Student students[size];

    for (int i = 0; i < size; i++) {
        scanf("%s %f", students[i].name, &students[i].score);
    }
    // Calculate mean
    float sum = 0;
    for (int i = 0; i < size; i++) {
        sum += students[i].score;
    }
    float mean = sum / size;
    // Calculate standard deviation
    float squared_diff_sum = 0;
    for (int i = 0; i < size; i++) {
        float diff = students[i].score - mean;
        squared_diff_sum += diff * diff;
    }
    float variance = squared_diff_sum / size;
    float std_dev = sqrt(variance);
    // Find student with highest and lowest scores
    struct Student *max_student = &students[0];
    struct Student *min_student = &students[0];
    for (int i = 1; i < size; i++) {
        if (students[i].score > max_student->score) {
            max_student = &students[i];
        }
        if (students[i].score < min_student->score) {
            min_student = &students[i];
        }
    }
    printf("%.2f %.2f %s %s\n", mean, std_dev, max_student->name, min_student->name);
    return 0;
}
```

โค้ดนี้ใช้โครงสร้าง struct Student เพื่อเก็บข้อมูลนักเรียนที่ประกอบด้วยชื่อและคะแนน จากนั้นทำการรับขนาดของกลุ่มนักเรียน และสร้างอาร์เรย์ของโครงสร้าง students ขนาดตามที่รับมาจากผู้ใช้ จากนั้นใช้ลูปเพื่อรับข้อมูลของแต่ละนักเรียนเข้าสู่อาร์เรย์ students หลังจากนั้นทำการคำนวณค่าเฉลี่ย (mean) และค่าเบี่ยงเบนมาตรฐาน (std\_dev) ของคะแนนทั้งหมด ส่วนถัดไปคือการหา้นักเรียนที่มีคะแนนสูงสุดและต่ำสุด โดยใช้ลูปเพื่อเปรียบเทียบคะแนนและระบุนักเรียนที่มีคะแนนสูงสุดและต่ำสุด ลูกท้ายนำผลลัพธ์ที่ได้มาแสดงผล ในรูปแบบที่กำหนดโดยแสดงค่าเฉลี่ย (mean) ค่าเบี่ยงเบนมาตรฐาน (std\_dev) และชื่อนักเรียนที่มีคะแนนสูงสุดและต่ำสุด

### Problem #2 | Jump Game

เกมกระโดด (Jump Game) ผู้เล่นจะเริ่มต้นที่ด้านแรก [0] และสามารถกระโดดได้ไกลสุดตามค่าที่ด้านนั้นบอก (เช่น ด้านที่ 5 กระโดดได้ 1-5) เป้าหมายคือ ผู้เล่นต้องกระโดดไปให้ถึงด้านสุดท้าย (ไม่เกินขอบเขต) โดยใช้จำนวนการกระโดด ถ้าทำได้ ให้แสดงจำนวนการกระโดด ถ้าไม่ ให้แสดง "Not Possible"

```
#include <limits.h>

int minJumps(int arr[], int n, int start, int end) {
    if (end == start) {
        return 0;
    }
    if (arr[start] == 0) {
        return INT_MAX;
    }

    int min = INT_MAX;
    for (int i = start + 1; i <= start + arr[start] && i <= end; i++) {
        int jumps = minJumps(arr, n, i, end);
        if (jumps != INT_MAX && jumps + 1 < min) {
            min = jumps + 1;
        }
    }

    return min;
}
```

#### ฟังก์ชัน minJumps:

- ตรวจสอบเงื่อนไขฐาน: ถ้า start เท่ากับ end ให้คืนค่า 0
- ถ้าระยะกระโดดที่ตำแหน่งเริ่มต้นเป็น 0 ให้คืนค่า INT\_MAX
- วนลูปหาตำแหน่งกระโดดถัดไปและเรียกช้าฟังก์ชัน minJumps เพื่อหาจำนวนครั้งน้อยที่สุด
- บันทึกค่าการกระโดดน้อยที่สุด (min)
- ส่งคืนค่าการกระโดดน้อยที่สุด (min)
- กรณีหากาเส้นทางไปไม่ถึง ส่งคืนค่า INT\_MAX

## Problem #3 | Symmetric Matrix

โจทย์นี้ให้ตรวจสอบเมทริกซ์ (ตารางตัวเลข) ว่าเป็นแบบสมมาตร (เท่ากันทุกแกนทแยง) หรือ แบบเบี้ยวสมมาตร (ลับเครื่องหมายทุกแกนทแยง) โดยใช้การสร้างและประมวลผลข้อมูลด้วยอาร์เรย์หลายมิติ (Multi-dimensional Array)

```
int main() {
    int m, n;

    if (scanf("%d %d", &m, &n) != 2) {
        fprintf(stderr, "Error: Invalid input\n");
        return 1;
    }
    int matrix[m][n];

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (scanf("%d", &matrix[i][j]) != 1) {
                fprintf(stderr, "Error: Invalid input\n");
                return 1;
            }
        }
    }
    int isSymmetric = 1, isSkewSymmetric = 1;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < i; j++) {
            if (matrix[i][j] != matrix[j][i]) {
                isSymmetric = 0;
            }
            if (matrix[i][j] != -matrix[j][i]) {
                isSkewSymmetric = 0;
            }
            if (!isSymmetric && !isSkewSymmetric) {
                break;
            }
        }
    }
    if (isSymmetric) {
        printf("The matrix is symmetric\n");
    } else if (isSkewSymmetric) {
        printf("The matrix is skew-symmetric\n");
    } else {
        printf("None\n");
    }
    return 0;
}
```

โดยล่วนที่สำคัญของ โค้ดจะมีดังนี้:

- การรับข้อมูล: โค้ดรับจำนวนแถว ( $m$ ) และจำนวนหลัก ( $n$ ) ของเมทริกซ์
- การสร้างอาร์เรย์: โค้ดสร้างอาร์เรย์  $matrix$  สองมิติเพื่อเก็บข้อมูลตัวเลขของเมทริกซ์
- การตรวจสอบ: โค้ดวนลูปตรวจสอบตัวเลขแต่ละตัวในเมทริกซ์
- ตรวจสอบว่าตัวเลขที่ตำแหน่ง  $matrix[i][j]$  เท่ากับตัวเลขที่ตำแหน่ง  $matrix[j][i]$  หรือไม่ (สำหรับแบบสมมาตร)
- ตรวจสอบว่าตัวเลขที่ตำแหน่ง  $matrix[i][j]$  เท่ากับ  $-1$  คูณตัวเลขที่ตำแหน่ง  $matrix[j][i]$  หรือไม่ (สำหรับแบบเบี้ยวสมมาตร)
- การแสดงผล: โค้ดแสดงผลลัพธ์ว่าเมทริกซ์เป็นแบบสมมาตรเบี้ยวสมมาตร หรือไม่ใช่ทั้งสองแบบ

## Problem #4 | Matrix Multiplication

โจทย์นี้เกี่ยวกับการทำการคูณเมทริกซ์ (Matrix Multiplication) โดยที่เราต้องนำความรู้เกี่ยวกับอาร์เรย์หลายมิติมาใช้ในการแก้ปัญหานี้ การคูณเมทริกซ์ เป็นการสร้างเมทริกซ์เดียวจากการคูณกันของสองเมทริกซ์

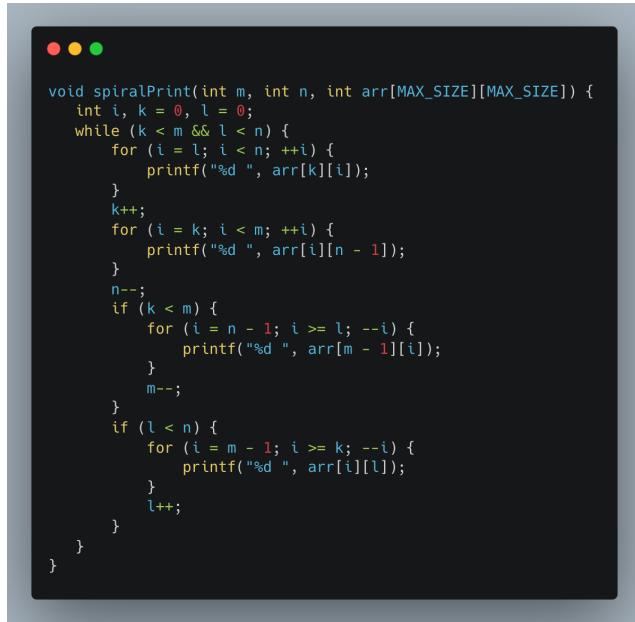
```
int **get_multiplication_matrix(int a_row, int a_col, int a[a_a_row][a_col], int b_row, int b_col, int b[b_b_row][b_col])
{
    int i, j, k;
    int **c = (int **)malloc(a_row * sizeof(int *));
    for (i = 0; i < a_row; i++)
    {
        c[i] = (int *)malloc(b_col * sizeof(int));
    }
    for (i = 0; i < a_row; i++)
    {
        for (j = 0; j < b_col; j++)
        {
            c[i][j] = 0;
            for (k = 0; k < a_col; k++)
            {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

โดยฟังก์ชันสำคัญของ โค้ดจะมีดังนี้:

ฟังก์ชัน `get_multiplication_matrix` ทำการคูณเมทริกซ์ A ขนาด  $a\_row \times a\_col$  กับเมทริกซ์ B ขนาด  $b\_row \times b\_col$  และสร้างเมทริกซ์ผลลัพธ์ที่เกิดจากการคูณโดยใช้การจองหน่วยความจำแบบ `dynamic memory allocation` และวนลูปเพื่อคำนวณค่าในเมทริกซ์ผลลัพธ์  $c$  ทุกรายการ และคืนค่าเป็น pointer ที่นำไปที่เมทริกซ์ผลลัพธ์ที่ได้

## Problem #5 | Spiral Array Printer

โจทย์ Spiral Array Printer นี้ เกี่ยวกับการรับข้อมูลและแสดงผลข้อมูลในลำดับที่เรียงเป็นวงกลม (Spiral Array) โดยให้ผู้ใช้ระบุขนาดและป้อนข้อมูลเพื่อแสดงผลลัพธ์ในรูปแบบที่เลื่อนตามลำดับเรียงทางวงกลมที่ระบุ ในโจทย์



```
void spiralPrint(int m, int n, int arr[MAX_SIZE][MAX_SIZE]) {  
    int i, k = 0, l = 0;  
    while (k < m && l < n) {  
        for (i = l; i < n; ++i) {  
            printf("%d ", arr[k][i]);  
        }  
        k++;  
        for (i = k; i < m; ++i) {  
            printf("%d ", arr[i][n - 1]);  
        }  
        n--;  
        if (k < m) {  
            for (i = n - 1; i >= l; --i) {  
                printf("%d ", arr[m - 1][i]);  
            }  
            m--;  
        }  
        if (l < n) {  
            for (i = m - 1; i >= k; --i) {  
                printf("%d ", arr[i][l]);  
            }  
            l++;  
        }  
    }  
}
```

- พังก์ชัน spiralPrint ใช้เทคนิคการเลื่อนตามลำดับทางวงกลมเพื่อพิมพ์ข้อมูลจากเมทริกซ์ arr ขนาด  $m \times n$  ในรูปแบบทางวงกลม โดยใช้ลูป while และลูป for เพื่อเลื่อนและพิมพ์ค่าข้อมูลทีละตำแหน่ง ตลอดจนครบถ้วนรายการในเมทริกซ์
- กำหนดตัวแปร k, l, m, และ n เพื่อกำหนดขอบเขตในการพิมพ์ในเมทริกซ์
  - ใช้ลูป while เพื่อเลื่อนทางวงกลมและตรวจสอบเงื่อนไข
  - ในลูป while, ใช้ลูป for และเงื่อนไขต่างๆ เพื่อพิมพ์ข้อมูลในลำดับทางวงกลม
  - ปรับค่าตัวแปร k, l, m, และ n ตามขั้นตอนการเลื่อนแต่ละขั้นตอน
  - ทำซ้ำขั้นตอน 2-4 จนกว่าทั้งสี่ขอบของเมทริกซ์จะ printf ครบ

# Week 3

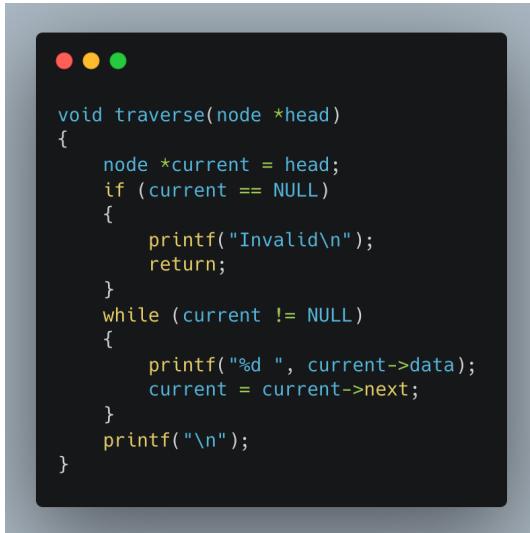
## Lab 3 Linked List

### Problem #1 | Linked Lists Insertion

โจทย์นี้เกี่ยวกับการสร้างฟังก์ชัน ในการแทรกข้อมูล ในลิงก์ลิสต์: การแทรกที่หัว (ที่เริ่มต้น) และที่ท้าย (สิ้นสุด) โดยรับค่าจำนวน โหนด ประเภทการแทรก (1 สำหรับดันและ 2 สำหรับท้าย) และโหนดต่าง ๆ ในลิงก์ลิสต์ ผลลัพธ์ แสดงลิงก์ลิสต์หลังจากการแทรกและ 'Invalid' ถ้าแสดงถ้าการแทรกไม่เป็นไปตามที่กำหนด



```
scanf("%d", &mode);
if (mode == 1)
{
    for (int i = 0; i < size; i++)
    {
        int data;
        scanf("%d", &data);
        node *new_node = (node *)malloc(sizeof(node));
        new_node->data = data;
        new_node->next = head;
        head = new_node;
    }
}
else if (mode == 2){
    for (int i = 0; i < size; i++)
    {
        int data;
        scanf("%d", &data);
        node *new_node = (node *)malloc(sizeof(node));
        new_node->data = data;
        new_node->next = NULL;
        if (head == NULL)
        {
            head = new_node;
        }
        else
        {
            node *current = head;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = new_node;
        }
    }
}
```



```
void traverse(node *head)
{
    node *current = head;
    if (current == NULL)
    {
        printf("Invalid\n");
        return;
    }
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

ในส่วนของโค้ด เริ่มต้นทำการรับขนาดของลิงก์ลิสต์ (size) และประเมินโหมดการแทรก (mode) จากนั้นทำการตรวจสอบว่าถ้าขนาดเป็น 0 จะแสดง "Invalid" และสิ้นสุดโปรแกรม ถ้าโหมดเป็น 1 จะทำการวนลูปเพื่อแทรกโหนดที่หัวของลิงก์ลิสต์ และถ้าโหมดเป็น 2 จะทำการวนลูปเพื่อแทรกโหนดที่ท้ายของลิงก์ลิสต์ โดยมีการสร้างโหนดใหม่และกำหนดค่าข้อมูลของโหนดในแต่ละลูป ทั้งนี้ โค้ดด้านในลูปจะต่างกันตามโหมด 1 และ 2 โดยโหมด 1 จะทำการแทรกที่หัวของลิงก์ลิสต์ และโหมด 2 จะทำการแทรกที่ท้ายของลิงก์ลิสต์ โดยมีการใช้ลูป while ในโหมด 2 เพื่อหาโหนดที่ว่างสุดของลิงก์ลิสต์ และทำการแทรกโหนดใหม่เข้าไปที่นั้น หลังจากนั้นจะทำการ traverse โดยการอ่านค่า next of node ไปเรื่อยๆ

## Problem #2 | Before or After Insertion ?

โจทย์นี้เกี่ยวกับการสร้างลิงก์ลิสต์จากโหนดที่กำหนดและการแทรกโหนดก่อนหรือหลังโหนดที่กำหนดตามคำสั่ง 'A r v' หรือ 'B r v' และ 'E' เพื่อลื้นสุดและแสดงลิงก์ลิสต์ที่สร้างขึ้น ผลลัพธ์คือลิงก์ลิสต์ที่ได้ โดยแสดงโหนดทั้งหมดและค้นด้วยชื่อว่าง

```
void insertAfter(node **head, int r, int v)
{
    node *newnode, *ptr, *preptr;
    newnode = (node *)malloc(sizeof(node));
    newnode->data = v;
    newnode->next = NULL;
    ptr = *head;
    preptr = ptr;
    while (preptr->data != r)
    {
        preptr = ptr;
        ptr = ptr->next;
    }

    if (r == (*head)->data)
    {
        newnode->next = (*head)->next;
        (*head)->next = newnode;
    }
    else
    {
        preptr->next = newnode;
        newnode->next = ptr;
    }
}

void insertBefore(node **head, int r, int v)
{
    node *new_node, *ptr, *preptr;
    new_node = (node *)malloc(sizeof(node));
    new_node->data = v;
    new_node->next = NULL;
    ptr = *head;
    preptr = ptr;
    while (ptr->data != r)
    {
        preptr = ptr;
        ptr = ptr->next;
    }

    if (r == (*head)->data)
    {
        new_node->next = *head;
        *head = new_node;
    }
    else
    {
        preptr->next = new_node;
        new_node->next = ptr;
    }
}
```

ส่วนหลักของโค้ด:

### insertAfter:

- สร้างโหนดใหม่ (newnode) และกำหนดค่าข้อมูลเป็น v และ next เป็น NULL
- ใช้ ptr และ preptr เพื่อวนลูปหาโหนดที่มีค่าเท่ากับ r ในลิงก์ลิสต์
- หาก r เป็นตัวแรกในลิงก์ลิสต์ จะแทรก newnode หลังจากตัวแรก
- หาก r ไม่ใช่ตัวแรก จะแทรก newnode ระหว่าง preptr และ ptr

### insertBefore:

- สร้างโหนดใหม่ (new\_node) และกำหนดค่าข้อมูลเป็น v และ next เป็น NULL
- ใช้ ptr และ preptr เพื่อวนลูปหาโหนดที่มีค่าเท่ากับ r ในลิงก์ลิสต์
- หาก r เป็นตัวแรกในลิงก์ลิสต์ จะแทรก new\_node ลงไปข้างหน้าตัวแรก
- หาก r ไม่ใช่ตัวแรก จะแทรก new\_node ระหว่าง preptr และ ptr

## Problem #3 | Where to DELETE ?

โจทย์นี้เกี่ยวกับการลบโหนดในลิงก์ลิสต์เดียวตามโหมดการลบที่กำหนด ได้แก่ 'F' ลบโหนดแรก, 'L' ลบโหนดท้าย, 'N x' ลบโหนดแรกที่มีค่า x, และ 'E' เพื่อลัญญาณลินสูตรการลบ หลังจากการลบ剩จลินให้พิมพ์โหนดที่เหลือในลิงก์ลิสต์ ถ้าไม่มีโหนดเหลือในลิงก์ลิสต์ให้พิมพ์ 'none' โดยจะมีฟังก์ชันหลักดังนี้

```
void deleteFirst(node **head)
{
    if (*head == NULL)
    {
        return;
    }
    node *temp = *head;
    *head = (*head)->next;
    free(temp);
}
void deleteLast(node **head)
{
    if (*head == NULL)
    {
        return;
    }
    node *ptr = *head;
    node *preptr = ptr;
    while (ptr->next != NULL)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = NULL;
    if (preptr == ptr)
    {
        *head = NULL;
    }
}
void deleteNumOnce(node **head, int num)
{
    node *ptr = *head;
    node *preptr = ptr;
    int is_found = 0;
    while (ptr != NULL)
    {
        preptr = ptr;
        ptr = ptr->next;
        if (ptr != NULL)
        {
            if (ptr->data == num)
            {
                is_found = 1;
                break;
            }
        }
    }
    if (is_found == 1)
    {
        if ((*head)->data == num)
        {
            deleteFirst(head);
        }
        else
        {
            node *temp = ptr;
            preptr->next = ptr->next;
            free(temp);
        }
    }
}
```

### deleteFirst:

- ตรวจสอบว่าลิงก์ลิสต์ว่างหรือไม่ ถ้าว่าง ให้กลับไปเลย
- สร้างตัวแปรชั่วคราว (**temp**) และกำหนดให้เป็นตัวแรกในลิงก์ลิสต์
- เลื่อนหัวลิงก์ลิสต์ไปยังโหนดถัดไป
- ลบโหนด **temp** ที่เป็นตัวแรก

### deleteLast:

- ตรวจสอบว่าลิงก์ลิสต์ว่างหรือไม่ ถ้าว่าง ให้กลับไปเลย
- ใช้ **ptr** และ **preptr** เพื่อวนลูปไปยังโหนดท้ายของลิงก์ลิสต์
- ตั้งค่า **preptr->next** เป็น **NULL** เพื่อลบโหนดท้าย
- ถ้า **preptr** และ **ptr** ชี้ไปที่โหนดเดียวกัน (หลังจากลบโหนดท้าย) ก็ ตั้งค่า **head** เป็น **NULL**

### deleteNumOnce:

- ใช้ **ptr** และ **preptr** เพื่อวนลูปลงมาในลิงก์ลิสต์เพื่อค้นหาโหนดที่มีค่าเท่ากับ **num**
- ถ้าพบ **num**, ตรวจสอบว่า **num** อยู่ที่ตัวแรกหรือไม่ หากใช่ให้ใช้ **deleteFirst** ในการลบ
- ถ้า **num** ไม่อยู่ที่ตัวแรก, ให้ลบโหนดที่พบ และเลื่อนการเชื่อมต่อของ **preptr** ไปยังโหนดถัดไป

## Problem #4 | Linked list that can go back

โจทย์นี้เกี่ยวกับการสร้างลิงก์ลิสต์ที่สามารถลิ้งค์ไปทั้งหน้าและหลัง ด้วยคำสั่ง ADD, DEL, SCH, และ END และแสดงผลลัพธ์ของลิงก์ลิสต์ที่สร้างขึ้นทั้งทิศทาง พังก์ชั่นสำคัญสำหรับโจทย์นี้หลักๆจะมี 3 พังก์ชั่นดังนี้:

```
void printListForward()
{
    struct Node *current = head;
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

### printListForward():

- ใช้เพื่อพิมพ์ค่าของโหนดในลิงก์ลิสต์ตามทิศทางปกติ (จากหัวไปท้าย)
- ใช้ลูป while เพื่อวนลูปตามลิงก์ลิสต์จนกว่าจะถึงสิ้นสุดของลิงก์ลิสต์
- แสดงค่าของโหนดและเลื่อนไปยังโหนดถัดไป

```
void printListBackward()
{
    struct Node *current = head;
    if (current == NULL)
    {
        return;
    }

    while (current->next != NULL)
    {
        current = current->next;
    }

    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->prev;
    }
    printf("\n");
}
```

### printListBackward():

- ใช้เพื่อพิมพ์ค่าของโหนดในลิงก์ลิสต์ตามทิศทางย้อนกลับ (จากหัวไปหัว)
- ใช้ลูป while เพื่อเลื่อนไปที่โหนดท้ายสุดของลิงก์ลิสต์
- ใช้ลูป while ถอยหลังเพื่อแสดงค่าของโหนดและย้อนกลับไปยังโหนดก่อนหน้าจนกว่าจะถึงหัวของลิงก์ลิสต์

```
void searchNeighbors(int data)
{
    struct Node *current = head;
    while (current != NULL && current->data != data)
    {
        current = current->next;
    }

    if (current == NULL)
    {
        printf("none\n");
    }
    else
    {
        if (current->prev == NULL)
        {
            printf("NULL ");
        }
        else
        {
            printf("%d ", current->prev->data);
        }

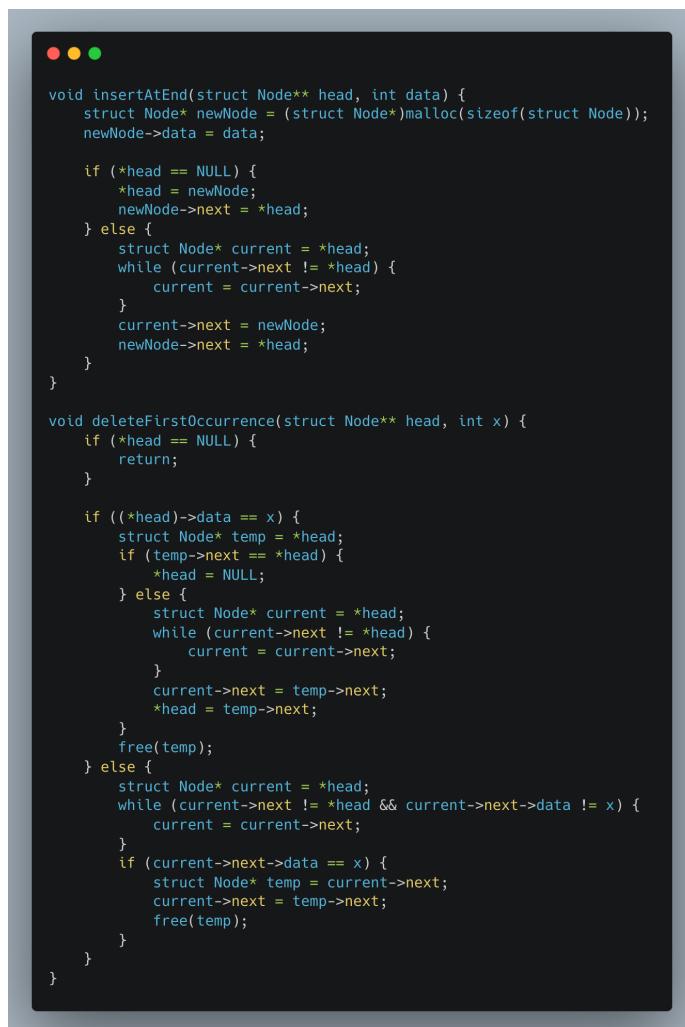
        if (current->next == NULL)
        {
            printf("NULL\n");
        }
        else
        {
            printf("%d\n", current->next->data);
        }
    }
}
```

### searchNeighbors(int data):

- ใช้เพื่อค้นหาโหนดที่มีค่าเท่ากับ data ในลิงก์ลิสต์
- ใช้ลูป while เพื่อวนลูปตามลิงก์ลิสต์จนกว่าจะพบโหนดที่มีค่าเท่ากับ data หรือจนกว่าจะถึงสิ้นสุดของลิงก์ลิสต์
- หากพบ, แสดงค่าของโหนดก่อนหน้าและต่อท้าย data นั้น โดยใช้ current->prev และ current->next
- ถ้าไม่พบ, แสดง "none"

## Problem #5 | Circular Linked List

เกี่ยวกับ Circular Linked List ที่คล้ายกับ Singly Linked List แต่มีการเปลี่ยนแปลงเล็กน้อยคือ โหนดสุดท้ายเชื่อมต่อกับหัวของลิงก์ลิสต์ ให้เขียนโปรแกรมและรับคำสั่ง 'I x' เพื่อแทรก x ลงใน โหนดสุดท้ายของ Circular Linked List และ 'D x' เพื่อลบโหนดแรกที่มีค่า x ถ้า x ไม่อยู่ใน Circular Linked List ให้ลับเว้น และ 'E' เพื่อสิ้นสุดโปรแกรม การเปลี่ยนแปลงและแสดงโหนดทั้งหมดในลิงก์ลิสต์ หากลิงก์ลิสต์ว่าง ให้แสดง "Empty" โดยจะมีฟังก์ชันหลักดังนี้



```
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head;
    } else {
        struct Node* current = *head;
        while (current->next != *head) {
            current = current->next;
        }
        current->next = newNode;
        newNode->next = *head;
    }
}

void deleteFirstOccurrence(struct Node** head, int x) {
    if (*head == NULL) {
        return;
    }

    if ((*head)->data == x) {
        struct Node* temp = *head;
        if (temp->next == *head) {
            *head = NULL;
        } else {
            struct Node* current = *head;
            while (current->next != *head) {
                current = current->next;
            }
            current->next = temp->next;
            *head = temp->next;
        }
        free(temp);
    } else {
        struct Node* current = *head;
        while (current->next != *head && current->next->data != x) {
            current = current->next;
        }
        if (current->next->data == x) {
            struct Node* temp = current->next;
            current->next = temp->next;
            free(temp);
        }
    }
}
}
```

### insertAtEnd():

- ใช้เพื่อแทรก โหนดใหม่ที่มีข้อมูลเท่ากับ data ที่สุดท้ายของ Circular Linked List
- สร้าง โหนดใหม่และกำหนดข้อมูลให้เท่ากับ data
- ถ้าลิงก์ลิสต์ว่าง, ให้โหนดใหม่เป็นหัวของลิงก์ลิสต์และเชื่อมต่อกับตัวเอง
- ถ้าไม่ใช่, หา โหนดสุดท้ายของลิงก์ลิสต์ และเชื่อมต่อกับ โหนดใหม่

### deleteFirstOccurrence():

- ใช้เพื่อลบ โหนดที่มีข้อมูลเท่ากับ x ครั้งแรกที่พบ ใน Circular Linked List
- ถ้าลิงก์ลิสต์ว่าง, ไม่ต้องทำอะไร
- ถ้า โหนดแรกมีข้อมูลเท่ากับ x, ลบ โหนดนั้น โดยตรวจสอบว่าลิงก์ลิสต์มี โหนดเดียวหรือไม่
  - ถ้าไม่ใช่, หา โหนดสุดท้ายของลิงก์ลิสต์ และเชื่อมต่อกับ โหนดตัดไปของ โหนดแรก และปรับหัวของลิงก์ลิสต์
  - ถ้า โหนดแรกไม่มีข้อมูลเท่ากับ x, ค้นหา โหนดที่มีข้อมูลเท่ากับ x ในลิงก์ลิสต์ และลบ โหนดนั้นถ้าพบ

## Assignment 3 Linked List

### Problem #1 | It's Sorting Time

สร้าง singly linked list เก็บรหัสนักศึกษาและคะแนน, เลือกโหมด "0" เรียงตามรหัส, โหมด "1" เรียงตามคะแนน (ถ้าคะแนนเท่ากันจะเรียงตามรหัส) และแสดงผลลัพธ์ของคะแนนที่เรียงตามโหมดที่เลือก โดยจะมีฟังก์ชันหลักดังนี้

```
void sortById(Node **head) {
    Node *current = *head, *next;
    while (current != NULL) {
        next = current->next;
        while (next != NULL) {
            if (current->id > next->id) {
                int tempId = current->id;
                int tempScore = current->score;
                current->id = next->id;
                current->score = next->score;
                next->id = tempId;
                next->score = tempScore;
            }
            next = next->next;
        }
        current = current->next;
    }
}

void sortByScore(Node **head) {
    Node *current = *head, *next;
    while (current != NULL) {
        next = current->next;
        while (next != NULL) {
            if (current->score > next->score) {
                int tempId = current->id;
                int tempScore = current->score;
                current->id = next->id;
                current->score = next->score;
                next->id = tempId;
                next->score = tempScore;
            } else if (current->score == next->score && current->id > next->id) {
                int tempId = current->id;
                int tempScore = current->score;
                current->id = next->id;
                current->score = next->score;
                next->id = tempId;
                next->score = tempScore;
            }
            next = next->next;
        }
        current = current->next;
    }
}
```

#### sortById:

- กำหนด current และ next ชี้ที่หัวของ linked list
- วนลูปหาข้อมูล โดยเปรียบเทียบและสลับตำแหน่งข้อมูลที่มีรหัสนักศึกษามากกว่า
- เลื่อน current และ next ไปตำแหน่งถัดไป
- ทำซ้ำขั้นตอน 2-3 จนไม่มีข้อมูลที่ต้องการสลับตำแหน่ง

#### sortByScore:

- กำหนด current และ next ชี้ที่หัวของ linked list
- วนลูปหาข้อมูล โดยเปรียบเทียบและสลับตำแหน่งข้อมูลที่คะแนนมากกว่าหรือเท่ากันและรหัสนักศึกษามากกว่า
- เลื่อน current และ next ไปตำแหน่งถัดไป
- ทำซ้ำขั้นตอน 2-3 จนไม่มีข้อมูลที่ต้องการสลับตำแหน่ง

### Problem #2 | Circular table

โต๊ะวงกลมมีจำนวน N ตัวเลขและขั้นตอนการลบตัวเลขเริ่มที่จุดที่กำหนดในวงกลม ให้หาตัวเลขสุดท้ายที่เหลืออยู่หลังจากการลบตัวเลขตามขั้นตอนที่กำหนด โดยจะมีฟังก์ชันหลักดังนี้

```
int findLastRemaining(int arr[], int size, int step) {
    int currentIndex = 0;
    while (size > 1) {
        currentIndex = (currentIndex + step - 1) % size;
        for (int i = currentIndex; i < size - 1; ++i) {
            arr[i] = arr[i + 1];
        }
        size--;
    }
    return arr[0];
}
```

`findLastRemaining` เป็นฟังก์ชันที่ใช้ในการหาตัวเลขที่เหลืออยู่ท้ายสุด ในอาร์เรย์หลังจากการลบตัวเลขตามขั้นตอนที่กำหนด. โดยใช้ลูป `while` เพื่อทำการลบตัวเลขและย้ายตำแหน่งของตัวเลข ในอาร์เรย์. การคำนวณดัชนีที่จะถูกลบทำได้โดยใช้สูตร  $(\text{currentIndex} + \text{step} - 1) \% \text{size}$ . ฟังก์ชันคืนค่าตัวเลขที่เหลือ

ท้ายสุด ในอาร์เรย์

## Problem #3 | Reverse Linked List

รับข้อมูล linked list และกลับด้านตัวเลข ในขอบเขต Sp ถึง Ep และแสดงผลลัพธ์ของ linked list ที่ถูกกลับด้านตามขอบเขตที่กำหนด โดยจะมีฟังก์ชันหลักดังนี้



```
ListNode* reverseBetween(ListNode* head, int m, int n) {
    if (head == NULL || m == n) {
        return head;
    }
    ListNode *dummy = malloc(sizeof(ListNode));
    dummy->next = head;
    ListNode *prev = dummy;
    for (int i = 1; i < m; i++) {
        prev = prev->next;
    }
    ListNode *current = prev->next;
    ListNode *next = NULL;
    for (int i = m; i < n; i++) {
        next = current->next;
        current->next = next->next;
        next->next = prev->next;
        prev->next = next;
    }
    return dummy->next;
}
```

reverseBetween ใช้เพื่อกลับด้าน linked list ระหว่างตำแหน่ง m ถึง n ใน linked list โดยใช้วิธีการสลับเปลี่ยนเชื่อมโยงของโหนดทั้งหมดในช่วงที่กำหนด:

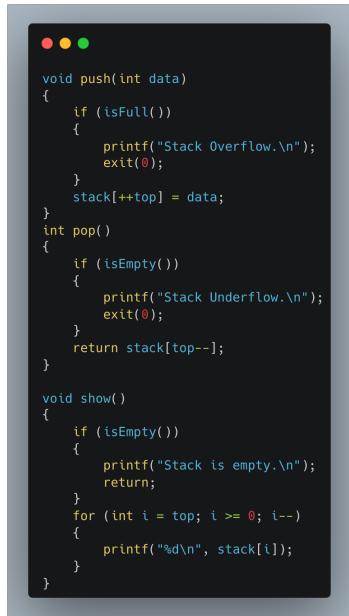
- ตรวจสอบว่า head เป็น NULL หรือ m เท่ากับ n จะคืนค่า head เดิมทันที
- สร้างโหนด dummy เพื่อจัดการกรณีพิเศษ และให้ prev เป็นตัวชี้ไปที่ dummy
- วนลูปหาตำแหน่งก่อนหน้าของ m และเก็บไว้ใน prev
- วนลูปกลับด้าน linked list ระหว่าง m ถึง n โดยใช้เทคนิคสลับเปลี่ยนโគงสร้างข้อมูล.
- คืนค่า dummy->next เพื่อให้ได้ linked list ที่ถูกกลับด้าน

# Week 4

## Lab 4 Stack

### Problem #1 | Stack Array

เป็นการสร้าง stack โดยใช้ array โดยมีการเขียนโปรแกรมที่ทำการดำเนินการต่างๆ เช่น push, pop, และแสดงค่าของ stack และแสดงข้อความต่างๆที่ปรากฏขึ้นในแต่ละฟังก์ชัน โดยจะมีฟังก์ชันหลักดังนี้



```
void push(int data)
{
    if (isFull())
    {
        printf("Stack Overflow.\n");
        exit(0);
    }
    stack[++top] = data;
}
int pop()
{
    if (isEmpty())
    {
        printf("Stack Underflow.\n");
        exit(0);
    }
    return stack[top--];
}

void show()
{
    if (isEmpty())
    {
        printf("Stack is empty.\n");
        return;
    }
    for (int i = top; i >= 0; i--)
    {
        printf("%d\n", stack[i]);
    }
}
```

- **push:** เพิ่มข้อมูลลงในสเต็ก ถ้าสเต็กเต็มจะแสดงข้อความ "Stack Overflow" และจบโปรแกรม มีการเพิ่มค่า data ในตำแหน่ง top แล้วเพิ่ม top ขึ้น 1
- **pop:** นำข้อมูลออกจากสเต็ก ถ้าสเต็กว่างจะแสดงข้อความ "Stack Underflow" และจบโปรแกรม นำค่าที่อยู่ที่ top ออกแล้วลดค่า top ลง 1
- **show:** แสดงข้อมูลในสเต็ก ถ้าสเต็กว่างจะแสดงข้อความ "Stack is empty" มีลูปที่วนโดยหลังเพื่อแสดงข้อมูลทุกรายการใน stack

### Problem #2 | Stack as linked list

เกี่ยวกับการใช้ stack พื้นฐาน สร้าง linked list ที่สามารถ push และ pop ได้ที่ตำแหน่งหน้าสุด พร้อมคำสั่ง 'o' เพื่อแสดงค่าที่ถูก pop, 't' เพื่อแสดงค่าบนสุด, 'e' เพื่อตรวจสอบว่าว่างหรือไม่, และ 'q' เพื่อจบโปรแกรม โดย x ต้องอยู่ในช่วง INT\_MIN ถึง INT\_MAX และแสดง "empty" เมื่อ stack ว่าง โดยจะมีฟังก์ชันหลักดังนี้



```
void push(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
}

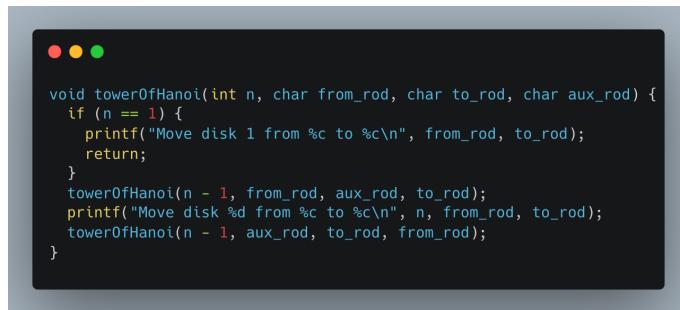
int pop() {
    if (head == NULL) {
        printf("empty\n");
        return -1;
    }
    Node *temp = head;
    int data = temp->data;
    head = head->next;
    free(temp);
    return data;
}

void top() {
    if (head == NULL) {
        printf("empty\n");
        return;
    }
    printf("%d\n", head->data);
}
```

- **push:** สร้างโหนดใหม่เพื่อเพิ่มข้อมูลลงใน linked list ที่เป็น stack และทำให้โหนดใหม่เป็นตัวหน้าของ linked list
- **pop:** ลบและคืนค่าข้อมูลของโหนดที่อยู่ด้านหน้าของ linked list ที่เป็น stack ถ้า linked list ว่างจะแสดง "empty" และคืนค่า -1
- **top:** แสดงค่าข้อมูลของโหนดที่อยู่ด้านหน้าของ linked list ที่เป็น stack ถ้า linked list ว่างจะแสดง "empty"

## Problem #3 | Tower of Hanoi

หาจำนวนในการย้าย disk ในหอคอยชานอย ให้น้อยที่สุด



```
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from %c to %c\n", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

### หลักการทำงาน:

- ถ้า  $n$  เป็น 1 (กรณีฐาน), ให้พิมพ์ข้อความ "Move disk 1 from {from\_rod} to {to\_rod}" ซึ่งหมายถึงการย้ายแผ่นดิสก์ 1 จากสาย {from\_rod} ไปยังสาย {to\_rod}
- ถ้า  $n$  ไม่ใช่ 1, ให้ทำการย้าย (recursively) แผ่นดิสก์  $n-1$  จากสาย {from\_rod} ไปยังสาย {aux\_rod} โดยใช้สาย {to\_rod} เป็นสายช่วย
- พิมพ์ข้อความ "Move disk {n} from {from\_rod} to {to\_rod}" ซึ่งหมายถึงการย้ายแผ่นดิสก์  $n$  จากสาย {from\_rod} ไปยังสาย {to\_rod}
- ทำการย้าย (recursively) แผ่นดิสก์  $n-1$  ที่อยู่บนสาย {aux\_rod} กลับไปยังสาย {to\_rod} โดยใช้สาย {from\_rod} เป็นสายช่วย

## Assignment 4 Stack

### Problem #1 | Ten to X

แปลงจำนวนเต็มบวกในฐานสิบเป็นจำนวนเต็มฐาน  $x$  โดยจะมีฟังก์ชันหลักดังนี้



```
void convertAndPrint(int n, int x) {
    if (n < 0 || x < 2 || x > 36) {
        printf("invalid\n");
        return;
    }
    Stack s;
    initStack(&s);
    while (n > 0) {
        int remainder = n % x;
        push(&s, remainder);
        n /= x;
    }
    while (!isEmpty(&s)) {
        int digit = pop(&s);
        if (digit < 10) {
            printf("%d", digit);
        } else {
            printf("%c", 'A' + digit - 10);
        }
    }
    printf("\n");
}
```

ฟังก์ชัน `convertAndPrint` ในโค้ดที่กำหนดมีลักษณะดังนี้:

- ตรวจสอบว่า  $n$  มีค่าน้อยกว่า 0 หรือ  $x$  ไม่ได้อยู่ในช่วงที่ถูกต้องระหว่าง 2 ถึง 36 หรือไม่ ถ้าเงื่อนไขเป็นจริงให้พิมพ์ "invalid" และสิ้นสุดฟังก์ชัน
- สร้าง `Stack` เพื่อเก็บผลลัพธ์ของการแปลง  $n$  จากฐาน 10 เป็นฐาน  $x$
- ใช้ลูป `while` เพื่อแปลง  $n$  เป็นฐาน  $x$  และนำเศษที่ได้มาใส่ใน `Stack`
- ในลูปถัดไป, นำข้อมูลจาก `Stack` มาพิมพ์ออกเป็นผลลัพธ์ของการแปลง โดยถ้าตัวเลขน้อยกว่า 10 ให้พิมพ์ตัวเลขตรงๆ ถ้ามีค่ามากกว่าหรือเท่ากับ 10 ให้แปลงเป็นตัวอักษร 'A' ถึง 'Z' และพิมพ์ผลลัพธ์

## Problem #2 | Palindrome checker



```
bool isPalindrome(char *input) {
    Stack s;
    initStack(&s);

    int length = strlen(input);

    for (int i = 0; i < length; i++) {
        push(&s, input[i]);
    }

    for (int i = 0; i < length; i++) {
        char popped = pop(&s);

        if (popped != input[i]) {
            return false;
        }
    }

    return true;
}
```

ฟังก์ชัน `isPalindrome` ในโค้ดที่กำหนดมีลักษณะดังนี้:

- สร้าง `Stack` เพื่อเก็บตัวอักษรจากข้อความ `input` ที่ต้องการตรวจสอบว่าเป็น `Palindrome` หรือไม่
- หาความยาวของข้อความ `input` โดยใช้ฟังก์ชัน `strlen` เพื่อกำหนดค่าให้กับตัวแปร `length`
- ในลูป `for` แรก, นำแต่ละตัวอักษรจาก `input` มาใส่ใน `Stack` โดยใช้ฟังก์ชัน `push`
- ในลูป `for` ที่สอง, นำตัวอักษรจาก `Stack` มาเปรียบเทียบกับตัวอักษรที่อยู่ใน `input` ที่ตำแหน่งปัจจุบัน ถ้าไม่ตรงกันให้ทำการ `return false`
- หากทุกรอบของลูปผ่านไปโดยไม่มีการ `return false` จะหมายถึง `input` เป็น `Palindrome` และจะ `return true`

## Problem #3 | Parenthesis Checker

ตรวจสอบว่าสตริงที่ได้รับมีวงเล็บที่เปิดและปิดสมดุลหรือไม่ โดยการใช้ `stack` เพื่อตรวจสอบความสมดุลของวงเล็บที่ปรากฏในสตริง. ถ้าวงเล็บทั้งหมดมีคู่ที่สมดุลกัน, สตริงจะถือว่าเป็น `balanced` โดยจะมีฟังก์ชันหลักดังนี้



```
bool isPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}

bool isBalanced(char *str) {
    Stack s;
    initStack(&s);

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '(' || str[i] == '{' || str[i] == '[') {
            push(&s, str[i]);
        } else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {
            if (isEmpty(&s) || !isPair(pop(&s), str[i])) {
                return false;
            }
        }
    }
    return isEmpty(&s);
}
```

`isPair` ใช้เพื่อตรวจสอบว่าวงเล็บที่เปิดและปิดสมดุลกัน

`isBalanced` ใช้ `stack` เพื่อตรวจสอบว่าสตริงที่รับมา มีวงเล็บที่เปิดและปิดสมดุล:

- สร้าง `stack` (`Stack s`)
- วนลูปผ่านทุกรอบของสตริง `str`
- ถ้าเจองวงเล็บเปิด, ให้นำลง `stack`
- ถ้าเจองวงเล็บปิด:
  - ถ้า `stack` ว่างหรือองเล็บปิดไม่สมดุลกัน, คืนค่า `false`
  - ถ้าไม่ว่าง, นำวงเล็บเปิดจาก `stack` และตรวจสอบว่าเป็นคู่กับวงเล็บปิด
- เมื่อทุกรอบถูกตรวจสอบและ `stack` ว่าง, คืนค่า `true`. ในกรณีอื่น, คืนค่า `false`

## Problem #4 | Infix to Postfix

เกี่ยวกับการแปลงนิพจน์ Infix เป็นนิพจน์ Postfix โดยรวมถึงตัวอักษร 'a-z' หรือ 'A-Z' เป็นสมาชิก และตัวดำเนินการ '+', '-', '\*', '/' ในนิพจน์ รวมทั้งวงเล็บ '(' เพื่อกำหนดความสำคัญของการดำเนินการ โดยให้ตอบนิพจน์ Postfix ที่ได้หลังจากการแปลง

```
● ● ●
void infixToPostfix(char *infix, char *postfix) {
    Stack s;
    initStack(&s);

    int i = 0;
    int j = 0;

    while (infix[i] != '\0') {
        char token = infix[i];

        if ((token >= 'a' && token <= 'z') || (token >= 'A' && token <= 'Z')) {
            postfix[j++] = token;
        } else if (token == '(') {
            push(&s, token);
        } else if (token == ')') {
            while (!isEmpty(&s) && s.items[s.top] != '(') {
                postfix[j++] = pop(&s);
            }
            pop(&s); // Pop the ')'
        } else {
            while (!isEmpty(&s) && getPrecedence(s.items[s.top]) >= getPrecedence(token)) {
                postfix[j++] = pop(&s);
            }
            push(&s, token);
        }
        i++;
    }
    while (!isEmpty(&s)) {
        postfix[j++] = pop(&s);
    }
    postfix[j] = '\0';
}
```

### infixToPostfix:

#### 1.วนลูปผ่านทุกตัวอักษรใน Infix:

- เพิ่มตัวแปรหรือตัวเลขลงใน Postfix
- ถ้าเป็นวงเล็บเปิด ให้ push ลงใน stack
- ถ้าเป็นวงเล็บปิด:

นำตัวอักษรออกจาก stack และเพิ่มลงใน Postfix

#### จนกว่าจะเจองวงเล็บเปิด

- ถ้าเป็นตัวดำเนินการ:
- ถ้า stack ไม่ว่าง และตัวดำเนินการบนสุดใน stack มีความสำคัญมากกว่าหรือเท่ากับตัวดำเนินการปัจจุบันให้ pop และเพิ่มลงใน Postfix

\* push ตัวดำเนินการปัจจุบันเข้า stack

#### 2.วนลูปท้ายเพื่อ pop ตัวดำเนินการที่เหลือใน stack และเพิ่มลงใน Postfix

#### 3.กำหนด \0 ที่ index สุดท้ายของ Postfix

# Week 5

## Lab 5 Queue

### Problem #1 | Spotify

ระบบจัดสรรเพลงที่ให้ผู้ใช้ทำการเพิ่มเพลง (enqueue), เล่นเพลงแรก ในคิว (dequeue), และแสดงรายการเพลง ทั้งหมดที่เหลือพร้อมระยะเวลาที่เหลือ (sum) ด้วยคำสั่งที่กำหนด โดยจะมีฟังก์ชันหลักๆดังนี้:

```
void enqueue(char name[], char artist[], int time)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    strcpy(newNode->data.name, name);
    strcpy(newNode->data.artist, artist);
    newNode->data.time = time;
    newNode->next = NULL;

    if (rear == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}

void dequeue()
{
    if (front == NULL)
    {
        printf("No songs in the playlist\n");
    }
    else
    {
        Node *temp = front;
        front = front->next;
        if (front == NULL)
        {
            rear = NULL;
        }
        printf("Now playing: %s by %s\n", temp->data.name, temp->data.artist);
        free(temp);
    }
}

void displayPlaylist()
{
    if (front == NULL)
    {
        printf("No songs in the playlist\n");
    }
    else
    {
        Node *temp = front;
        int sum = 0;

        printf("Songs in the playlist:\n");
        while (temp != NULL)
        {
            printf("%s by %s\n", temp->data.name, temp->data.artist, temp->data.time);
            sum += temp->data.time;
            temp = temp->next;
        }
        printf("Remaining Time: %d\n", sum);
    }
}
```

**enqueue:** เพิ่มเพลงลง ในรายการเล่น โดยสร้างโหนดใหม่และ เชื่อมต่อกับรายการเล่นที่มีอยู่แล้ว หากรายการว่างเปล่า กำหนด front และ rear ให้ชี้ที่โหนดใหม่

**dequeue:** เล่นเพลงที่อยู่ที่ด้านหน้าของรายการเล่น ถ้ารายการว่างเปล่า แสดงข้อความ "No songs in the playlist" มีฉะนั้น ลบ โหนดที่เก็บข้อมูลของเพลงนั้น และปรับ front ให้ชี้ที่เพลงถัดไป หากหลังการเปลี่ยนนี้ front เป็น NULL ปรับ rear เป็น NULL

**displayPlaylist:** แสดงข้อมูลทั้งหมดในรายการเล่น ถ้ารายการว่างเปล่า แสดงข้อความ "No songs in the playlist" มีฉะนั้น วนลูป ผ่านโหนดแต่ละตัวแสดงชื่อเพลง, ศิลปิน, และระยะเวลา พร้อมคำนวณและแสดงผลรวมเวลาของเพลงทั้งหมดในรายการ

### Problem #2 | เมื่อไหร่จะถึงคิวจันบ้าง???

คิววงกลมสำหรับคลินิกทันตกรรม ใช้ insertq, dequeue, และ show ตามคำสั่ง 'I', 'D', 'S', 'E'. คำสั่งทำงานตาม FIFO

```
int circularQueue[10];
int front = -1, rear = -1;
void insertq(int value) {
    if ((front == 0 && rear == size - 1) || (front == rear + 1) || (front == -1 && rear == size - 1)) {
        printf("Queue is full!!\n");
        return;
    }
    if (front == -1) {
        front = 0;
        rear = 0;
    } else {
        if (rear == size - 1)
            rear = 0;
        else
            rear = rear + 1;
    }
    circularQueue[rear] = value;
}
int dequeue() {
    int deletedValue;
    if (front == -1) {
        printf("Queue is empty!!\n");
        return -1;
    }
    deletedValue = circularQueue[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        if (front == size - 1)
            front = 0;
        else
            front = front + 1;
    }
    return deletedValue;
}
```

#### Insert (insertq):

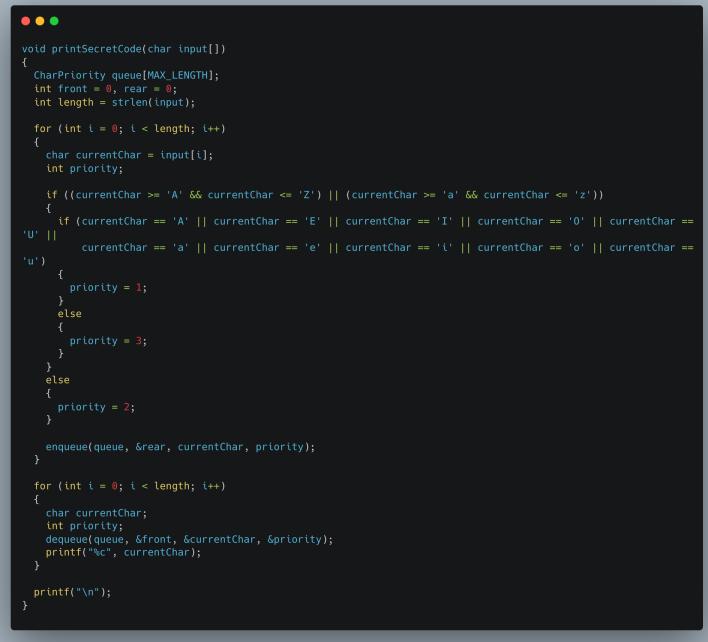
- ตรวจสอบคิวว่าเต็มหรือไม่
- กำหนดตำแหน่งเริ่มต้นในกรณีคิวว่าง
- ในกรณี rear ไปถึงตำแหน่งสุดท้าย ให้กลับไปที่ตำแหน่งแรก
- เพิ่มค่าในตำแหน่งที่ rear ชี้ไป

#### Dequeue:

- ตรวจสอบคิวว่าว่างหรือไม่
- เก็บค่าที่จะถูกลบ
- ในกรณี front และ rear ชี้ไปที่ตำแหน่งเดียวกัน (คิวมีข้อมูลเพียงตัวเดียว)
- ในกรณี front ไปถึงตำแหน่งสุดท้าย ให้กลับไปที่ตำแหน่งแรก
- เพิ่มค่าของ front เพื่อเลื่อนไปยังตำแหน่งถัดไป

## Problem #3 | Secret Code Only You and I Know

แปลงสติงตามกฎเรียงลำดับความสำคัญของตัวอักษรในภาษาอังกฤษ, โดยให้พิจารณาพยัญชนะมีความสำคัญสูงสุด, ตามด้วยสระ, และตัวอักษรอื่น ๆ ส่วนตัวอักษรที่มีความสำคัญเท่ากันจะถูกเรียงตามลำดับที่ป้อนเข้ามา



```
void printSecretCode(char input[])
{
    CharPriority queue[MAX_LENGTH];
    int front = 0, rear = 0;
    int length = strlen(input);

    for (int i = 0; i < length; i++)
    {
        char currentChar = input[i];
        int priority;

        if ((currentChar >= 'A' && currentChar <= 'Z') || (currentChar >= 'a' && currentChar <= 'z'))
        {
            if (currentChar == 'A' || currentChar == 'E' || currentChar == 'I' || currentChar == 'O' || currentChar == 'U')
                currentChar == 'a' || currentChar == 'e' || currentChar == 'i' || currentChar == 'o' || currentChar == 'u'
            {
                priority = 1;
            }
            else
            {
                priority = 3;
            }
            else
            {
                priority = 2;
            }
            enqueue(queue, &rear, currentChar, priority);
        }
    }

    for (int i = 0; i < length; i++)
    {
        char currentChar;
        int priority;
        dequeue(queue, &front, &currentChar, &priority);
        printf("%c", currentChar);
    }

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

### สร้าง Queue:

- สร้าง array queue และตัวแปร front, rear เพื่อแทนตำแหน่งแรกและสุดท้ายของ queue
- กำหนดความยาวของสติงเป็น length ด้วย strlen

### Loop และกำหนด Priority:

- วนลูปผ่านทุกตัวอักษรในสติง
- ดึงตัวอักษรและกำหนด priority ตามกฎที่กำหนด
- เพิ่มตัวอักษรและ priority ใน queue ด้วย enqueue

### Loop และแสดงผลลัพธ์:

- วนลูปผ่านตัวอักษรใน queue
- ดึงตัวอักษรและแสดงผล

### แสดงผลลัพธ์:

แสดงขึ้นบรรทัดใหม่ท้ายสุด

# Week 6

## Active Learning 1 Report

### Problem : การจัดลำดับคนขึ้นชิงช้าสวรรค์

#### Introduction

ชิงช้าสวรรค์เป็นเครื่องเล่นยอดนิยม ในสวนสนุกและงานรื่นเริงต่างๆ การจัดลำดับผู้เล่นขึ้นชิงช้าสวรรค์อย่างมีประสิทธิภาพจะเป็นสิ่งสำคัญเพื่อลดเวลาการรอคิว เพิ่มความพึงพอใจของผู้เล่น และความปลอดภัยในการใช้งาน การจัดลำดับคนขึ้นชิงช้าสวรรค์แบบดั้งเดิมมักใช้พังก์งานโดยจัดการทำให้เกิดปัญหาหลายประการ เช่น ความล่าช้า ความผิดพลาด ความไม่โปร่งใส ความเสี่ยงด้านความปลอดภัย การใช้โครงสร้างข้อมูล (data structure) จะช่วยจัดลำดับคนขึ้นชิงช้าสวรรค์ได้อย่างมีประสิทธิภาพ โดยแก้ปัญหาต่างๆ ให้ดีขึ้น

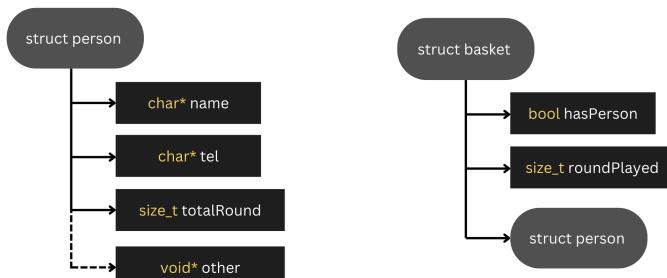
#### Relevant Theories

การใช้ทฤษฎีที่เกี่ยวข้องกับโครงสร้างข้อมูลเชิงเส้นมีความสำคัญในการแก้ไขปัญหาของการจัดลำดับคนขึ้นชิงช้าสวรรค์ โครงสร้างข้อมูลที่สามารถนำมาใช้ได้มีดังนี้:

- **Array:** เก็บข้อมูลต่อเนื่องแบบสูตรได้เร็ว แต่มีขนาดคงที่ที่ต้องกำหนดที่เริ่มต้น.
- **Linked List:** เก็บข้อมูลใน node และมีการเชื่อมต่อกัน, สามารถเพิ่มหรือลบข้อมูลได้โดยไม่ต้องกำหนดขนาด, แต่การเข้าถึงข้อมูลต้องทำการเคลื่อนที่ผ่าน node.
- **Stack:** ใช้เพื่อจัดเก็บข้อมูลแบบ Last-In, First-Out (LIFO) และมักใช้ในการทำงานเกี่ยวกับการเรียกฟังก์ชัน.
- **Queue:** ใช้เพื่อจัดเก็บข้อมูลแบบ First-In, First-Out (FIFO) และเหมาะสมสำหรับการจัดลำดับคนขึ้นชิงช้าสวรรค์

#### Methodology

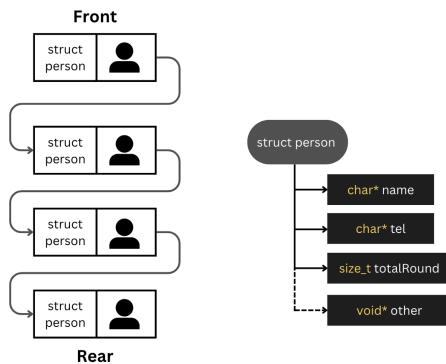
ในรายงานนี้กลุ่มเราได้เลือกใช้เป็น Simple Queue มาจัดการในเรื่องการต่อคิวและ Circular linked list มาจัดการในเรื่องการนับรอบในการหมุนของชิงช้า เนื่องจาก Simple Queue ทำตามหลัก FIFO เพื่อลดความล่าช้า และ Circular linked list ทำการค้นหาข้อมูลได้อย่างรวดเร็ว โดยโครงสร้างข้อมูลที่นำมาใช้จะมีอยู่ 2 อย่างคือ person สำหรับเก็บข้อมูลผู้คนที่อยู่ในคิว และ basket สำหรับกระเช้าในชิงช้าสวรรค์โดยในรายงานนี้กำหนดให้ 1 basket บรรจุได้ 1 คนดังรูป



( ภาพที่ 1 โครงสร้างข้อมูล )

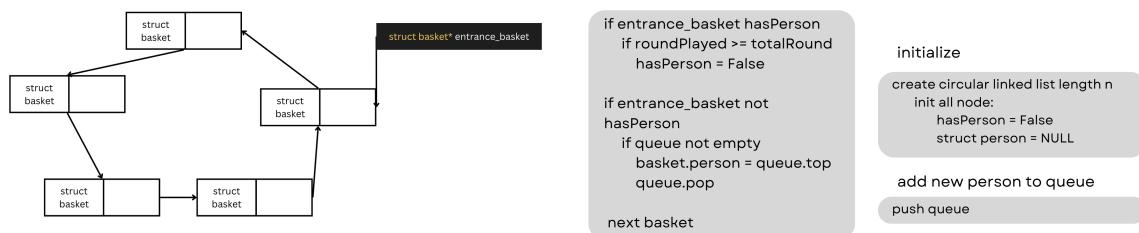
การนำโครงสร้างข้อมูลดังกล่าวไปใช้ในการจัดลำดับขึ้นชิงช้าสวรรค์จะแบ่งเป็น 2 ขั้นตอนดังนี้:

1. การต่อคิว ก่อนขึ้นชิงเข้าสวรรค์: นำ person ไปจัดคิวใน simple queue เพื่อต่อແກວตามลำดับดังรูป



## ( ภาพที่ 2 การต่อคิว ก่อนขึ้นชิงช้าสวรรค์ )

2. การจัดสรรคนชั้นกระเบื้องช้าสวรรค์: ก่อนนำคนชั้นกระเบื้องเข้าเริ่มต้นจะทำการเช็คก่อนว่ามีคนอยู่ในกระเบื้องแล้วหรือไม่จาก basket->hasPerson หากมีคนอยู่แล้วให้เช็คว่าคนในกระเบื้องชั้นนี้มีคนครบจำนวนรอบที่ซื้อไว้หรือยังจาก basket->roundPlayed หากครบแล้วให้นำคนที่อยู่ในกระเบื้องชั้นนี้ออกและนำคนที่อยู่บน top ของคิวเข้ามาแทนและเช็ค basket ถัดไปเปรียบเทียบเมื่อมีการหมุนของชิงชาดังรูป



( ภาพที่ 3.1 Circular Linked List ของชิงช้าสวรรค์)

( ภาพที่ 3.2 Sudocode คร่าวๆ)