

Work_Log_2024_07_10

C언어 8일차 과제

작성자 : 박기홍

Remind

Remind

1. 로빏에 입단한 후 저는 군사용 로봇과 구조용 로봇을 개발하여 사람들의 안전과 생명을 지키는 데 기여하는 것을 목표로 임하겠습니다.
2. 지도교수님과 선배님들을 공경하며 동기들을 존중하는 마음가짐으로 임하겠습니다.
3. 로빏에서 배운 공학적 기술과 경험은 사람들을 살리는 데 사용될 수 있으며, 악한 마음을 품게 된다면 사람들에게 피해를 줄 수 있다는 것을 명심하면서 올바른 로봇 엔지니어로서 성장할 수 있도록 윤리를 중요시하겠습니다.
4. 글로벌 로봇 산업의 성장기와 성숙기를 이끌어 나아가는 엘리트로 성장할 수 있는 능력을 만들겠습니다.

TODO List

Info

어떤 것을 할지 미리 생각해 놓는 시간입니다. 간략하게 2~5개로 적습니다.

- ☒ 옴사디언 생성하기
- ☐ 과제-1 하기
- ☒ 과제-2 하기
- ☒ 과제-3 하기
- ☒ 과제-4 하기

Activity

Info

오늘 작업한 내용을 작성합니다.

- 개발을 하면서 고민한 부분
- 개발을 하면서 참고한 문서
- 새로 알게 된 사실 등의 내용

1. 오피디언 생성하기

Start Time : 20:05

End Time : 20:06

2. 과제-1 하기

Start Time : 21:00

End Time : 21:30

3. 과제-2 하기

Start Time : 2024.07.11. 14:00

End Time : 2024.07.11. 17:30

3. 과제-3 하기

Start Time : 2024.07.11. 18:30

End Time : 2024.07.11. 17:00

2. 과제-4 하기

Start Time : 21:30 / 2024.07.11. 00:00

End Time : 22:00 / 2024.07.11. 05:30

Today's Completion

 Info

오늘 완료한 작업을 정리합니다.

1. 오피디언 생성하기

2. 과제-2 하기

3. 과제-3 하기

4. 과제-4 하기

Today's Concept

포인터 함수

스택

큐

포인터 변수가 구조체 내부에 있을 경우

Solving a HomeWork

 Info

과제 풀이를 작성합니다.

HW_001

 과제 설명

단순 연결 리스트를 구현하기.

- insert : 원하는 위치에 node 추가(그 전 data와 index모두 가능하도록)
- insert_back : 연결리스트의 맨 끝에 node 추가
- insert_first : 연결리스트의 맨 처음에 node 추가
- delete : 원하는 요소 삭제(원하는 요소는 data와 index모두 가능하도록)
- delete_first : 연결리스트 맨 처음 node 삭제
- delete_back : 연결리스트 맨 마지막 node 삭제
- get_entry : 요소 찾기(data로 찾을 시 index 반환, index로 찾을 시 data 반환)
- get_length : 리스트 전체 길이 반환
- print_list : 리스트의 모든 요소 출력
- reverse : 리스트 역순으로 만들기
- Data 자료형은 자유

[머리말]

지난 번에는 어렵고, 이해가 안 되어 포기했던 과제를 이번에는 70% 넘게 구현해 낼 수 있었습니다. 개인적으로 과제를 제한 시간내에 완수하지 못하여, 일부 기능만 구현된 소스코드를 보내드립니다.

[소스 코드]

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct _Node {
    int data;
    struct _Node* next;
}Node;

typedef struct _LinkedList {
    Node* head;
    Node* cur;
    Node* tail;
```

```

    int* data[100];
    int size;
}LinkedList;

void Initializing_Node(LinkedList* list);
void Inserting_Node(LinkedList* list, int num, int value);
void Inserting_Node_Back(LinkedList* list, int value);
void Inserting_Node_First(LinkedList* list, int value);
void Deleting_Node_First(LinkedList* list, int value);
void Printing_List(LinkedList* list);

int main() {
    // LinkedList 선언 및 동적할당함.
    LinkedList* linkedList = (LinkedList*)malloc(sizeof(LinkedList));
    Initializing_Node(linkedList);

    char* inputText = (char*)malloc(sizeof(char) * 20);
    printf("[SYSTEM]노드가 초기화 되었습니다.\n");
    while (1){
        printf("\n\n[SYSTEM]사용 가능한 명령어 모음\n\n");
        printf("\t1. insert\t2.insert_back\t3.insert_first\n");
        printf("\t4. delete\t5.delete_first\t6.delete_back\n");
        printf("\t7. get_length\t8.print_list\t9.reverse\n\n");
        printf("[SYSTEM]명령을 입력하세요 : ");
        scanf("%s", inputText);

        if(strcmp(inputText, "insert") == 0){
            int numLocation, nodeValue;
            printf("[SYSTEM]원하는 위치를 입력하세요. 위치 선택 (%d ~ %d) : ", 0,
linkedList->size);
            scanf("%d", &numLocation);
            if (0 <= numLocation && numLocation <= linkedList->size) {
                printf("[SYSTEM]원하는 노드의 값을 입력하세요. : ");
                scanf("%d", &nodeValue);
                Inserting_Node(linkedList, numLocation, nodeValue);
            }else{
                printf("[SYSTEM]정해진 범위 내에서 생성해야 합니다.\n\n");
            }
        }
        else if (strcmp(inputText, "insert_back") == 0) {
            int nodeValue;
            printf("[SYSTEM]원하는 노드의 값을 입력하세요. : ");
            scanf("%d", &nodeValue);
            Inserting_Node_Back(linkedList, nodeValue);
        }else if (strcmp(inputText, "insert_first") == 0) {
            int nodeValue;
            printf("[SYSTEM]원하는 노드의 값을 입력하세요. : ");
            scanf("%d", &nodeValue);
            Inserting_Node_First(linkedList, nodeValue);
        }else if (strcmp(inputText, "delete") == 0) {

        }else if (strcmp(inputText, "delete_first") == 0) {

        }else if (strcmp(inputText, "delete_back") == 0) {

```

```

        }else if (strcmp(inputText, "get_entry") == 0) {

        }else if (strcmp(inputText, "get_length") == 0) {

        }else if (strcmp(inputText, "print_list") == 0) {
            Printing_List(linkedList);
        }else if (strcmp(inputText, "reverse") == 0) {

        }

    }

    return 0;
}

void Initializing_Node(LinkedList* list) {
    // 노드들의 앞 부분인 head는 NULL로 초기화 함.
    list->head = NULL;
    list->cur = NULL;
    list->tail = NULL;
    // 사용하는 노드가 없으므로, 노드의 집합인 list의 size 값도 0으로 초기화 함.
    list->size = 0;
}

void Inserting_Node(LinkedList* list, int num, int value) {
    // Node 선언 및 동적할당함.
    Node* node = (Node*)malloc(sizeof(node));

    // 노드 추가하기.
    node->data = value; // 노드의 값은 입력 받은 value로 지정하기.
    list->size++;

    // Debugging:
    printf("\n\n현재 노드의 수 : %d개, 입력된 값 : %d\n\n", list->size, node->data);
}

void Inserting_Node_Back(LinkedList* list, int value) {

    list->size++;
    if (0 < list->size) { // stack의 용량 범위 이내라면 push를 진행함.
        Node* node = (Node*)malloc(sizeof(node));

        node->data = value;
        node->next = NULL;

        list->head = node;

        list->data[list->size - 1] = node->data;
    }
}

```

```

void Inserting_Node_First(LinkedList* list, int value) {
    list->size++;
    if (0 < list->size) { // stack의 용량 범위 이내라면 push를 진행함.
        Node* node = (Node*)malloc(sizeof(node));

        node->data = value;
        node->next = NULL;

        list->head = node;

        for (int i = 0; i < list->size - 1; i++) {
            list->data[list->size - i - 1] = list->data[list->size - i - 2];
        }
        list->data[0] = node->data;
    }
}

void Printing_List(LinkedList* list) {

    printf("NULL");
    for (int i = 0; i < list->size; i++) {
        int value = list->data[i];
        printf("<-%d", value);
    }
}

```

[테스트 케이스]

Test Case#1

```

C:\Users\Worlde\Github\WROE
[SYSTEM]노드가 초기화 되었습니다.

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_back
[SYSTEM]원하는 노드의 값을 입력하세요. : 1

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_back
[SYSTEM]원하는 노드의 값을 입력하세요. : 2

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_back
[SYSTEM]원하는 노드의 값을 입력하세요. : 3

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : print_list
NULL<-1<-2<-3

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : |

```

Test Case#2

```
C:\Users\Wlordk\WGithub\WROE x + v
[SYSTEM]노드가 초기화 되었습니다.

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_back
[SYSTEM]원하는 노드의 값을 입력하세요. : 1

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_back
[SYSTEM]원하는 노드의 값을 입력하세요. : 3

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : insert_first
[SYSTEM]원하는 노드의 값을 입력하세요. : 99

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : print_list
NULL<-99<-1<-3

[SYSTEM]사용 가능한 명령어 모음
1. insert      2.insert_back  3.insert_first
4. delete      5.delete_first 6.delete_back
7. get_length  8.print_list  9.reverse

[SYSTEM]명령을 입력하세요 : |
```

HW_002

과제 설명

단순 연결 리스트를 이용한 stack구현하기.

- push : 정수 push
- pop : pop하고 pop된 값 출력. stack이 비어있을 시 비어있다고 출력
- size : stack 크기 출력
- top : top에 위치한 값 반환.
- isEmpty : stack에 데이터가 없으면 true, 있으면 false 반환
- printStack : stack 내 모든 값 출력. stack이 비어있을 시 비어있다고 출력
- Data 자료형 자유

[소스 코드]

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```

typedef struct _Node {
    int data;
    struct _Node* next;
}Node;

typedef struct _Stack {
    Node* top;
    Node* cur;
    int* data[100];
    int size;
}Stack;

void InitializingStack(Stack* tStack); // Stack 초기화 함수.
void PushingStack(Stack* tStack, int num);
void PoppingStack(Stack* tStack);
void Printing_Size(Stack* tStack);
int Getting_Top(Stack* tStack);
int Checking_isEmpty(Stack* tStack);
void Printing_List(Stack* tStack);

int main() {

    // Stack 선언 및 동적할당함.
    Stack* stack = (Stack*)malloc(sizeof(stack));

    char* inputText = (char*)malloc(sizeof(char) * 20);
    InitializingStack(stack);
    printf("[SYSTEM]스택이 초기화 되었습니다.\n");

    while (1){
        printf("\n\n[SYSTEM]사용 가능한 명령어 모음\n\n");
        printf("\t1. push\t2.pop\t 3.size\n");
        printf("\t4. top\t5.isEmpty 6.printStack\n\n");
        printf("[SYSTEM]명령을 입력하세요 : ");
        scanf("%s", inputText);

        int tempNum;
        if(strcmp(inputText, "push") == 0){
            printf("[SYSTEM]push할 값을 입력하세요 : ");
            scanf("%d", &tempNum);
            PushingStack(stack, tempNum); // stack은 Call by ref로, tempNum은
Call by Value로 보냄.
        }else if (strcmp(inputText, "pop") == 0) {
            PoppingStack(stack);
        }else if (strcmp(inputText, "size") == 0) {
            Printing_Size(stack);
        }else if (strcmp(inputText, "top") == 0) {
            if (Getting_Top(stack) == 0) {
                printf("현재 top에 위치한 값은 없습니다.\n\n");
            }else{

```



```

        printf("현재 top에 위치한 값은 %d입니다.\n\n",
Getting_Top(stack));
    }
    }else if (strcmp(inputText, "isEmpty") == 0) {
        if (Checking_isEmpty(stack) == 1) {
            printf("현재 Stack에 데이터가 없는 상태 입니다.\n\n");
        }else{
            printf("현재 Stack에 데이터가 있는 상태 입니다.\n\n");
        }

        }else if (strcmp(inputText, "printStack") == 0) {
            Printing_List(stack);
        }
    }

}

void InitializingStack(Stack* tStack) {

    tStack->size = 0;
    tStack->top = -1; // Stack의 top의 초기 상태는 -1 상태임.
    tStack->cur = NULL;
}

void PushingStack(Stack* tStack, int num) {

    tStack->size++;
    if (0 < tStack->size) { // stack의 용량 범위 이내라면 push를 진행함.
        Node* node = (Node*)malloc(sizeof(node));

        node->data = num;
        node->next = NULL;

        /*tStack->top++;*/
        tStack->top = node;
        tStack->cur = node;

        tStack->data[tStack->size] = node->data;
    }
}

void PoppingStack(Stack* tStack) {

    if (tStack->size == 0) {
        printf("Stack이 비었습니다.\n");
    }else{
        printf("%d이(가) pop되었습니다.\n\n", tStack->data[tStack->size]);
        tStack->data[tStack->size] = NULL;
        tStack->size--;
    }
}

```

```

void Printing_Size(Stack* tStack) {
    printf("스택의 size는 %d입니다.\n\n", tStack->size);
}

int Getting_Top(Stack* tStack) {
    if (tStack->size == 0) {
        return 0; // top에 노드가 없는 상태 일때는 0을 반환하여 정해진 문구를 출력함.
    }else{
        return tStack->data[tStack->size]; // top에 위치한 값 반환하기.
    }
}

int Checking_isEmpty(Stack* tStack) {

    //true값과 false값을 반환하라고 하였으므로, 각각 1과 0을 반환함.
    if(tStack->size == 0){
        return 1;
    }else{
        return 0;
    }
}

void Printing_List(Stack* tStack) {

    if (tStack->size == 0) {
        printf("현재 Stack에 data가 없습니다.\n\n");
    }else{
        printf("[   스택   ]\n");
        printf("i-----i\n");
        for (int i = 0; i < tStack->size; i++) {
            printf("I %5d   I\n", tStack->data[tStack->size - i]);
        }
        printf("I-----I");
    }
}

```

[테스트 케이스]

Test Case#1

```
C:\Users\WordkW\Github\WROE x + v
[SYSTEM]스택이 초기화 되었습니다.

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : push
[SYSTEM]push할 값을 입력하세요 : 10

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : printStack
[ 스택 ]
i-----i
I  10  I
I-----I

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : |
```

Test Case#2

```
C:\Users\WordkW\Github\WROE x + v

4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : push
[SYSTEM]push할 값을 입력하세요 : 20

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : push
[SYSTEM]push할 값을 입력하세요 : 30

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : pop
30이(가) pop되었습니다.

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : printStack
[ 스택 ]
i-----i
I  20  I
I  10  I
I-----I

[SYSTEM]사용 가능한 명령어 모음

1. push 2.pop 3.size
4. top 5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : |
```

Test Case#3

```
C:\Users\Wlordk\WGithub\WROE x + v
[SYSTEM]명령을 입력하세요 : printStack
[   스택   ]
i-----i
I   20   I
I   10   I
I-----I

[SYSTEM]사용 가능한 명령어 모음

      1. push 2.pop      3.size
      4. top  5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : size
스택의 size는 2입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. push 2.pop      3.size
      4. top  5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : top
현재 top에 위치한 값은 20입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. push 2.pop      3.size
      4. top  5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : isEmpty
현재 Stack에 데이터가 있는 상태 입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. push 2.pop      3.size
      4. top  5.isEmpty 6.printStack

[SYSTEM]명령을 입력하세요 : |
```

HW_003

과제 설명

단순 연결 리스트를 이용한 Queue 구현

- Enqueue : Queue에 data 입력
- Dequeue : Dequeue하고 Dequeue된 값 출력. Queue가 비어있을 시 비어있다고 출력
- size : Queue 크기 출력
- front : front에 위치한 값 반환.
- rear : rear에 위치한 값 반환.
- isEmpty : Queue에 데이터가 없으면 true, 있으면 false 반환
- printQueue : Queue 내 모든 값 출력. Queue가 비어있을 시 비어있다고 출력

[소스 코드]

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```

typedef struct _Node {
    int data;
    struct _Node* next;
}Node;

typedef struct _Queue {
    Node* front;
    Node* cur;
    int* data[100];
    int size;
}Queue;

void InitializingQueue(Queue* sQueue); // Queue 초기화 함수.
void Enqueueing(Queue* sQueue, int num);
void Dequeueing(Queue* sQueue);
void Printing_Size(Queue* sQueue);
int Getting_Front(Queue* sQueue);
int Getting_Rear(Queue* sQueue);
int Checking_isEmpty(Queue* sQueue);
void Printing_List(Queue* sQueue);

int main() {

    // Queue 선언 및 동적할당함.
    Queue* queue = (Queue*)malloc(sizeof(queue));

    char* inputText = (char*)malloc(sizeof(char) * 20);
    InitializingQueue(queue);
    printf("[SYSTEM]큐가 초기화 되었습니다.\n");

    while (1){
        printf("\n\n[SYSTEM]사용 가능한 명령어 모음\n\n");
        printf("\t1. Enqueue\t2. Dequeue\t 3.size\n");
        printf("\t4. front\t5.rear\t\t 6.isEmpty\n");
        printf("\t7. printQueue\n\n");
        printf("[SYSTEM]명령을 입력하세요 : ");
        scanf("%s", inputText);

        int tempNum;
        if(strcmp(inputText, "Enqueue") == 0){
            printf("[SYSTEM]Enqueue할 값을 입력하세요 : ");
            scanf("%d", &tempNum);
            Enqueueing(queue, tempNum);
        }else if (strcmp(inputText, "Dequeue") == 0) {
            Dequeueing(queue);
        }else if (strcmp(inputText, "size") == 0) {
            Printing_Size(queue);
        }else if (strcmp(inputText, "front") == 0) {
            if (Getting_Front(queue) != NULL) {
                printf("front에 위치한 값은 %d입니다.\n\n",

```

```

    Getting_Front(queue));
        }else{
            printf("front에 위치한 값이 없습니다.\n\n");
        }
    }else if (strcmp(inputText, "rear") == 0) {
        if (Getting_Rear(queue) != NULL) {
            printf("rear에 위치한 값은 %d입니다.\n\n",
Getting_Rear(queue));
        }else{
            printf("rear에 위치한 값이 없습니다.\n\n");
        }
    }else if (strcmp(inputText, "isEmpty") == 0) {
        if (Checking_isEmpty(queue) == 1) {
            printf("Queue에 데이터가 있습니다.\n\n");
        }else{
            printf("Queue에 데이터가 없습니다.\n\n");
        }
    }else if (strcmp(inputText, "printQueue") == 0) {
        Printing_List(queue);
    }
}

}

}

void InitializingQueue(Queue* sQueue) {

    sQueue->size = 0;
    sQueue->front = 0; // Queue의 초기 상태는 0임.
    sQueue->cur = NULL;
}

void Enqueueing(Queue* sQueue, int num) {

    sQueue->size++;

    if (0 < sQueue->size) { // 큐의 용량 범위 내라면 Enqueue를 진행함.
        Node* node = (Node*)malloc(sizeof(Node));

        node->data = num;
        node->next = NULL;

        sQueue->front = node;
        sQueue->data[sQueue->size - 1] = node->data;
    }
}

void Dequeueing(Queue* sQueue) {

    if (sQueue->size == 0) {
        printf("Queue가 비었습니다.\n\n");
    }else{

```

```

        printf("%d이(가) Dequeue 되었습니다.\n\n", sQueue->data[0]);

        for (int i = 0; i < sQueue->size - 1; i++) {
            sQueue->data[i] = sQueue->data[i + 1];
        }
        sQueue->size--;
    }
}

void Printing_Size(Queue* sQueue) {
    printf("큐의 size는 %d입니다.\n\n", sQueue->size);
}

int Getting_Front(Queue* sQueue) {
    if (0 < sQueue->size) {
        return sQueue->data[0];
    }else{
        return NULL;
    }
}

int Getting_Rear(Queue* sQueue) {
    if (0 != sQueue->size) {
        return sQueue->data[sQueue->size - 1];
    }else{
        return NULL;
    }
}

int Checking_isEmpty(Queue* sQueue) {
    if (sQueue->size != 0) {
        return 1;
    }else{
        return 0;
    }
}

void Printing_List(Queue* sQueue) {
    if (sQueue->size == 0) {
        printf("현재 Queue에 data가 없습니다.\n\n");
    }else {
        printf("[   큐   ]\n");
        printf("i-----i\n");
        for (int i = 0; i < sQueue->size; i++) {
            printf("I %5d   I\n", sQueue->data[i]);
        }
    }
}

```

```

        printf("I-----I");
    }
}

```

[테스트 케이스]

Test Case#1

```

C:\Users\Wlord\W\Github\WROE x + v
[SYSTEM]큐가 초기화 되었습니다.

[SYSTEM]사용 가능한 명령어 모음
      1. Enqueue    2. Dequeue    3. size
      4. front      5. rear       6. isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : Enqueue
[SYSTEM]Enqueue할 값을 입력하세요 : 10

[SYSTEM]사용 가능한 명령어 모음
      1. Enqueue    2. Dequeue    3. size
      4. front      5. rear       6. isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : Enqueue
[SYSTEM]Enqueue할 값을 입력하세요 : 20

[SYSTEM]사용 가능한 명령어 모음
      1. Enqueue    2. Dequeue    3. size
      4. front      5. rear       6. isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : Enqueue
[SYSTEM]Enqueue할 값을 입력하세요 : 30

[SYSTEM]사용 가능한 명령어 모음
      1. Enqueue    2. Dequeue    3. size
      4. front      5. rear       6. isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : printQueue
[   큐   ]
I-----I
I  10   I
I  20   I
I  30   I
I-----I

[SYSTEM]사용 가능한 명령어 모음
      1. Enqueue    2. Dequeue    3. size
      4. front      5. rear       6. isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : |

```

Test Case#2


```
C:\Users\WordkW\Github\WROE x + v
[SYSTEM]Enqueue할 값을 입력하세요 : 30

[SYSTEM]사용 가능한 명령어 모음

1. Enqueue      2.Dequeue      3.size
4. front        5.rear        6.isEmpty
7. printQueue

[SYSTEM]명령을 입력하세요 : printQueue
[   큐   ]
i-----i
I   10   I
I   20   I
I   30   I
I-----I

[SYSTEM]사용 가능한 명령어 모음

1. Enqueue      2.Dequeue      3.size
4. front        5.rear        6.isEmpty
7. printQueue

[SYSTEM]명령을 입력하세요 : Dequeue
10이(가) Dequeue 되었습니다.

[SYSTEM]사용 가능한 명령어 모음

1. Enqueue      2.Dequeue      3.size
4. front        5.rear        6.isEmpty
7. printQueue

[SYSTEM]명령을 입력하세요 : Enqueue
[SYSTEM]Enqueue할 값을 입력하세요 : 90

[SYSTEM]사용 가능한 명령어 모음

1. Enqueue      2.Dequeue      3.size
4. front        5.rear        6.isEmpty
7. printQueue

[SYSTEM]명령을 입력하세요 : printQueue
[   큐   ]
i-----i
I   20   I
I   30   I
I   90   I
I-----I

[SYSTEM]사용 가능한 명령어 모음

1. Enqueue      2.Dequeue      3.size
4. front        5.rear        6.isEmpty
7. printQueue

[SYSTEM]명령을 입력하세요 : |
```

Test Case#3

```
C:\Users\WordkW\Github\WROE x + v
[SYSTEM]명령을 입력하세요 : printQueue
[   큐   ]
i-----i
I   20   I
I   30   I
I   90   I
I-----I

[SYSTEM]사용 가능한 명령어 모음

      1. Enqueue      2.Dequeue      3.size
      4. front        5.rear         6.isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : size
큐의 size는 3입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. Enqueue      2.Dequeue      3.size
      4. front        5.rear         6.isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : front
front에 위치한 값은 20입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. Enqueue      2.Dequeue      3.size
      4. front        5.rear         6.isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : rear
rear에 위치한 값은 90입니다.

[SYSTEM]사용 가능한 명령어 모음

      1. Enqueue      2.Dequeue      3.size
      4. front        5.rear         6.isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : isEmpty
Queue에 데이터가 있습니다.

[SYSTEM]사용 가능한 명령어 모음

      1. Enqueue      2.Dequeue      3.size
      4. front        5.rear         6.isEmpty
      7. printQueue

[SYSTEM]명령을 입력하세요 : |
```

HW_004

과제 설명

회문 판별하기

- 회문 또는 팰린드롬(Palindraome)은 앞 뒤 방향으로 볼 때 같은 순서의 문자로 구성된 문자열을 말한다.
- Ex) abba, madam, was it a cat i saw
- 문자열을 입력받아 회문인지 아닌지 판별하여 출력
- hint :
 - 스택과 큐를 이용(LIFO, FIFO)

[소스 코드]

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

// Stack 구조체 정의하기.
typedef struct _textStack {
    int max; // stack의 용량 저장. (동적)
```

```

    char ptr; // stack의 포인터.
    char qPtr; // stack에서 Deque를 하기 위한 포인터. Queue 전용 Struct를 만드는 것 말고, 동
일한 Struct에서 Deque가 되도록 시도해 봄.
    char* stk[]; // stack을 가리키는 포인터.
}TextStack;

// 필요한 사용자 정의 함수를 프로토타입으로 선언하기.
void InitializingStack(TextStack* tStack, char* text, int* textCnt); // Stack 초기화 함수.
void PushingStack(TextStack* tStack, char* text); // Stack에 push하는 함수.
void PoppingStack(TextStack* tStack, char* text); // Stack에 pop하는 함수.
/* Stack에서 Deque하는 함수.
* 원래는 Queue에서 Deque를 해야 하지만,
* 미리 선언한 Stack에 Deque 기능을 추가하는 것으로 설계함.
*/
void DequeueingStack(TextStack* tStack, char* text);

int main() {

    // 단어, 문장이 입력 되며 회문 여부를 스택과 큐 개념을 사용하여 구해야 함.
    // 1. 문자열을 동적 할당으로 받아 봄시다.
    char* inputText = NULL;
    inputText = (char*)malloc(sizeof(char) * 100);

    TextStack* textStack = (TextStack*)malloc(sizeof(TextStack));

    // inputText의 총 갯수를 저장하기 위한 변수.
    int temp = 0;
    int* textCnt = &temp;

    // 형식 입출력하기.
    printf("회문 여부를 파악하고 싶은 문자열 입력 : ");
    scanf("%[^\n]s", inputText);

    // Stack 초기화하기.
    InitializingStack(textStack, inputText, textCnt);

    // inputText 전체를 Stack에 Push하기.
    PushingStack(textStack, inputText);

    textStack->ptr = textStack->max - 1;
    /* 회문 판별하기.
    * for문은 *textCnt / 2 만큼만 돌리면 됨.
    * why?)
    * 1. inputText의 문자 수가 홀수면 가운데 숫자를 제외한 나머지 수가 똑같은지 비교하면 됨. (중
복 참조 방지)
    * 2. inputText의 문자 수가 짝수면 가운데 숫자가 없으므로, *textCnt / 2 만큼 돌리면 서로 중
복되지 않게 비교할 수 있음.
    */
    int isPalindrom = 1; // 회문임을 나타내는 boolean 대체 변수.
    for (int i = 0; i < *textCnt / 2; i++) {
        if (textStack->stk[textStack->qPtr] != textStack->stk[textStack->ptr]) {

```

```

        isPalindrom = 0;
    }
    // Stack에서 Dequeue하기 = 앞에서 하나 빼기.
    DequeueingStack(textStack, inputText);
    // Stack에서 Pop하기 = 뒤에서 하나 빼기.
    PoppingStack(textStack, inputText);
}

if (isPalindrom) {
    printf("\n\n\t\t\t\t\t%s은(는) 회문입니다.\n\n", inputText);
} else {
    printf("\n\n\t\t\t\t\t%s은(는) 회문이 아닙니다.\n\n", inputText);
}

// 동적 메모리 할당 해제 하기.
free(inputText);

return 0;
}

void InitializingStack(TextStack* tStack, char* text, int* textCnt) {

    int cnt = 0;
    for (int i = 0; *(text + i) != NULL; i++) {
        // 공백으로 문자열을 읽어 들였지만, 공백을 제외한 문자의 수만 저장함. -> pop과
deque를 원활하게 하기 위함임.
        if (*(text + i) != 32) {
            cnt++;
        }
    }
    tStack->ptr = 0; // stack 초기화 시, stack의 pointer는 0으로 초기화 함.
    tStack->qPtr = 0; // stack 초기화 시, queue의 pointer는 0으로 초기화 함.
    tStack->max = cnt; // stack의 용량은 입력된 문자열의 문자 수로 지정함.
    *textCnt = cnt;
}

void PushingStack(TextStack* tStack, char* text) {
    if (tStack->ptr < tStack->max) { // stack의 용량 범위 이내라면 push를 진행함.
        for (int i = 0; *(text + i) != NULL; i++) {
            // 공백을 제외한 모든 문자를 push함. -> pop과 deque를 원활하게 하기 위함
임.
            if (*(text + i) != 32) {
                tStack->stk[tStack->ptr] = *(text + i);
                tStack->ptr++;
            }
        }
    }
}

void PoppingStack(TextStack* tStack, char* text) { // 하나씩 pop 하기.
    /*tStack->ptr = tStack->max - 1;*/
    if (tStack->qPtr < tStack->ptr) { // stack의 용량 범위 이내라면 push를 진행함.
        tStack->stk[tStack->ptr] = NULL;
        tStack->max--;
        tStack->ptr--;
    }
}

```

```

    }
}

void DequeueingStack(TextStack* tStack, char* text) { // 하나씩 deque 하기.
    if (tStack->qPtr < tStack->max) { // stack의 용량 범위 이내라면 push를 진행함.
        tStack->stk[tStack->qPtr] = NULL;
        tStack->qPtr++;
    }
}
}

```

[테스트 케이스]

Test Case#1

Microsoft Visual Studio 디버그

회문 여부를 파악하고 싶은 문자열 입력 : abba

abba은(는) 회문입니다.

C:\Users\lordk\Github\ROBIT_Intern_KiHongPark_HW_repo\Projects\C_Language\Homework\HW_20240710\x64\Debug\HW_20240710.exe (프로세스 3064개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

Test Case#2

Microsoft Visual Studio 디버그

회문 여부를 파악하고 싶은 문자열 입력 : was it a cat i saw

was it a cat i saw은(는) 회문입니다.

C:\Users\lordk\Github\ROBIT_Intern_KiHongPark_HW_repo\Projects\C_Language\Homework\HW_20240710\x64\Debug\HW_20240710.exe (프로세스 33052개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

Test Case#3

```
Microsoft Visual Studio 디버그 × + -
회문 여부를 파악하고 싶은 문자열 입력 : I wanna be a member of ROBIT!

I wanna be a member of ROBIT!은(는) 회문이 아닙니다.

C:\Users\lordk\Github\ROBIT_Intern_KiHongPark_HW_repo\Projects\C_Language\Homework\HW_20240710\x64\Debug\HW_20240710.exe
(프로세스 31332개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Ideas & Important Information

 Info

아이디어 및 중요 정보를 작성합니다.

Memo

 Info

작업 중 기타 내용을 메모합니다.

Review

 Info

하루 작업에 대한 피로도, 기분 등을 평가합니다.

Feelings : (😞)

Fatigue : (5)

Summary : ~~~