과제 1 소스코드

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

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

typedef struct _LinkedList
{
    Node* head;
    int size;
} LinkedList;

void List(LinkedList* list);

void insert(LinkedList* list, int data, int index);

void insert_back(LinkedList* list, int data);

void insert_first(LinkedList* list, int data);

void delete_first(LinkedList* list);

void delete_back(LinkedList* list);

int get_entry(LinkedList* list, int data);

int get_length(LinkedList* list);

void print_list(LinkedList* list);


int main()
{
    int data;
    char* input;
    LinkedList list;
    List(&list);
   /* while (1) scanf를 사용해 문자를 입력받아 작동하는 프로그램을 만드려 했으나
오류로 실패
```

```
{
scanf_s("[^₩n] %d", input, data, index);
if (strcmp(input, "insert_first"))
{
    insert_first(&list, data);
    print_list;
}
else if (strcmp(input, "insert"))
{
    insert(&list, data, index);
    print_list;
}
else if (strcmp(input, "insert_back"))
{
    insert_back(&list, data);
    print_list;
}
else if (strcmp(input, "delete_back"))
{
    delete_back(&list);
    print_list;
}
else if (strcmp(input, "delete_first"))
{
    delete_first(&list);
    print_list;
}
else if (strcmp(input, "get_entry"))
{
    int result;
    result = get_entry(&list, data);
    printf("주소는 : %d", result);
}
else if (strcmp(input, "get_length"))
{
    int result;
    result = get_length(&list);
    printf("길이는 : %d", result);
}
}*/

insert_first(&list, 100);

print_list(&list);
```

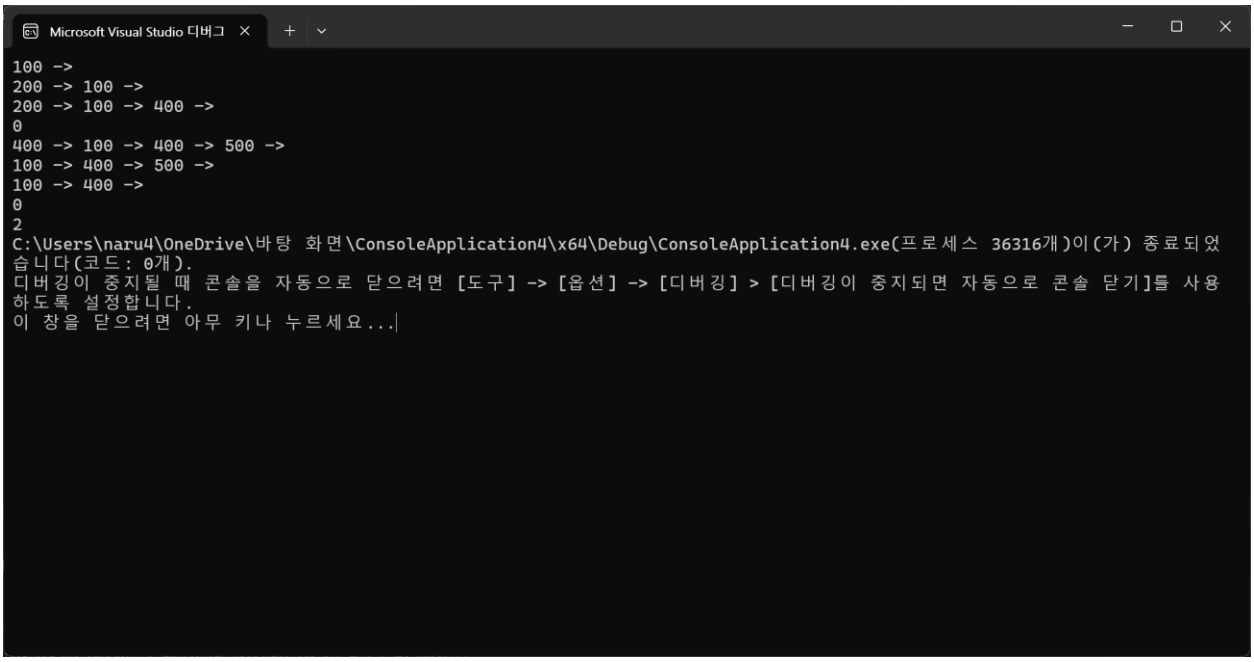
```
    insert_first(&list, 200);

    print_list(&list);

    insert(&list, 400, 2);

    print_list(&list);

    printf("%d₩n", get_entry(&list, 400));//400의 주소 여기서는 존재 o

    insert_back(&list, 500);

    print_list(&list);

    delete_first(&list);

    print_list(&list);

    delete_back(&list);

    print_list(&list);

    printf("%d₩n", get_entry(&list, 400));//400의 주소 여기서는 존재 x

    printf("%d", get_length(&list));

    return 0;
}

void List(LinkedList* list)
{
    list->head = NULL;//시작
    list->size = 0;
}

void insert(LinkedList* list, int data, int index)
{

    Node* newNode = (Node*)malloc(sizeof(Node));//새로운 노드 동적할당
    newNode->data = data;//새로운 노드의 데이터 영역에 데이터 값 입력

    if (index == 0) //새로운 노드의 주소가 0이라면
    {
```

```
        newNode->next = list->head;//노드의 다음을 전의 헤드로 정하고
        list->head = newNode;//헤드를 새로운 노드로 지정
    }
    else
    {
        Node* current = list->head;
        for (int i = 0; i < index - 1; ++i) //새로운 노드의 주소의 전번까지 current를
내림
        {
            current = current->next;
        }
        //current는 추가하려는 주소의 전
        newNode->next = current->next;//새로운 노드의 next를 current의 다음 노드에
연결
        current->next = newNode;//current의 노드를 원래 다음 노드가 아닌 새로운
노드에 연결
    }
    list->size++;//리스트의 크기+1
}

void insert_back(LinkedList* list, int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;//새로운 노드의 다음은 없음

    if (list->head == NULL) //아무 노드도 없을때
    {
        list->head = newNode;//새로운 노드가 헤드
    }
    else
    {
        Node* current = list->head;//current는 헤드에서 시작
        while (current->next) //다음 노드가 있을때까지 current를 낮춤,next가
연결되있지 않은 노드가 나오면 끝남
        {
            current = current->next;
        }
        current->next = newNode;//tail노드의 next를 새로운 노드로 연결
    }
    list->size++;//사이즈 추가
}

void insert_first(LinkedList* list, int data)
```

```
{
    Node* newNode = (Node*)malloc(sizeof(Node));//새로운 노드 동적할당
    newNode->data = data;//데이터 입력
    newNode->next = list->head;//새로운 노드 다음은 전 헤드
    list->head = newNode;//배열의 헤드로 지정
    list->size++;//사이즈+1
}

void delete_first(LinkedList* list)
{
    if (list->head == NULL) //노드가 없을때
    {
        printf("노드가 존재하지 않습니다₩n");
        return;
    }
    Node* temp = list->head;//헤드 노드를 temp라는 주소를 붙여줌
    //만약 free(list->head)로 바로 동적 할당 해제 시키면 헤드 다음 노드를 헤드로
지정할 방법이 없음
    list->head = list->head->next;//헤드 노드의 다음을 헤드 노드로 지정
    free(temp);//헤드 노드 동적 할당 해제
    list->size--;//사이즈 감소
}

void delete_back(LinkedList* list)
{
    if (list->head == NULL)
    {
        printf("노드가 존재하지 않습니다₩n");
        return;
    }
    if (list->head->next == NULL)//헤드 노드 하나만 있을때
    {
        free(list->head);//헤드 동적할당 해제
        list->head = NULL;//헤드 부분은 공백으로 지정
    }
    else
    {
        Node* current = list->head;
        while (current->next->next != NULL)//현재의 ->next ->next가 null 이라면
current->next가 tail, 따라서 tail전까지 이동
        {
            current = current->next;
        }
        free(current->next);//tail 동적 할당 해제
```

```
        current->next = NULL;//tail전 노드의 다음은 NULL
    }
    list->size--;
}

int get_entry(LinkedList* list, int data)
{
    Node* current = list->head;
    int index = 0;//인덱스라는 변수(주소)
    while (current != NULL)
    {
        if (current->data = data)//current 노드의 데이터가 입력된 숫자와 동일이라면
        {
            return index;//index 주소 반환
        }
        else
        {
            current = current->next;
            index++;//주소 +1
        }
    }
}

int get_length(LinkedList* list)
{
    return list->size;//사이즈 반환
}

void print_list(LinkedList* list)
{
    Node* current = list->head;
    while (current != NULL)//헤드부터 테일까지(NULL로 이어지지 않을때까지)
    {
        printf("%d -> ", current->data);//current 노드의 데이터 출력
        current = current->next;//current가 다음 노드로
    }
    printf("₩n");
}
```

실행결과



Delete와 reverse는 구현 실패

과제 2 소스코드

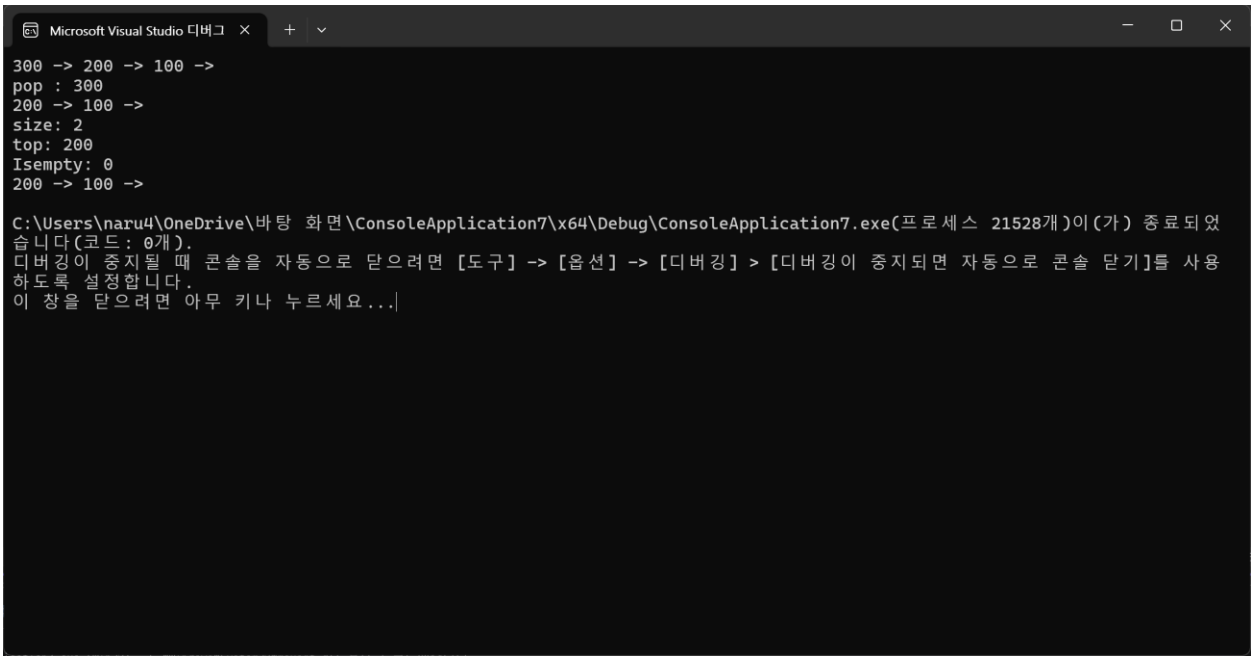
```c
#include <stdio.h>
#include <stdlib.h>

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

typedef struct _Stack
{
    Node* top;
    int size;
} Stack;

void _Stack(Stack* stack);
void push(Stack* stack, int data);
int pop(Stack* stack);
int size(Stack* stack);
int top(Stack* stack);
int isEmpty(Stack* stack);
```

```
void printStack(Stack* stack);

int main()
{
    Stack stack;
    _Stack(&stack);

    push(&stack, 100);
    push(&stack, 200);
    push(&stack, 300);

    printStack(&stack);

    printf("pop : %d₩n", pop(&stack));
    printStack(&stack);

    printf("size: %d₩n", size(&stack));
    printf("top: %d₩n", top(&stack));
    printf("Isempty: %d₩n", isEmpty(&stack));

    printStack(&stack);

    return 0;
}

void _Stack(Stack* stack)
{
    stack->top = NULL;
    stack->size = 0;
}

void push(Stack* stack, int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = stack->top;
    stack->top = newNode;
    stack->size++;
}

int pop(Stack* stack)
{
    if (stack->top == NULL)
    {
```

```
        printf("스택이 비어있습니다₩n");
        return 0;
    }
    Node* temp= stack->top;
    int data = temp->data;
    stack->top = stack->top->next;
    free(temp);
    stack->size--;
    return data;
}

int size(Stack* stack)
{
    return stack->size;
}

int top(Stack* stack)
{
    if (stack->top == NULL)
    {
        printf("스택이 비어있습니다₩n");
        return 0;
    }
    return stack->top->data;
}

int isEmpty(Stack* stack)
{
    if (stack->size = 0)
        return 1;
    else
        return 0;
}

void printStack(Stack* stack)
{
    if (stack->top == NULL)
    {
        printf("스택이 비어있습니다₩n");
        return;
    }
    Node* current = stack->top;
    while (current != NULL)
    {
```

```
        printf("%d -> ", current->data);
        current = current->next;
    }
Printf("₩n");
}
```

실행 결과



과제 3 소스코드

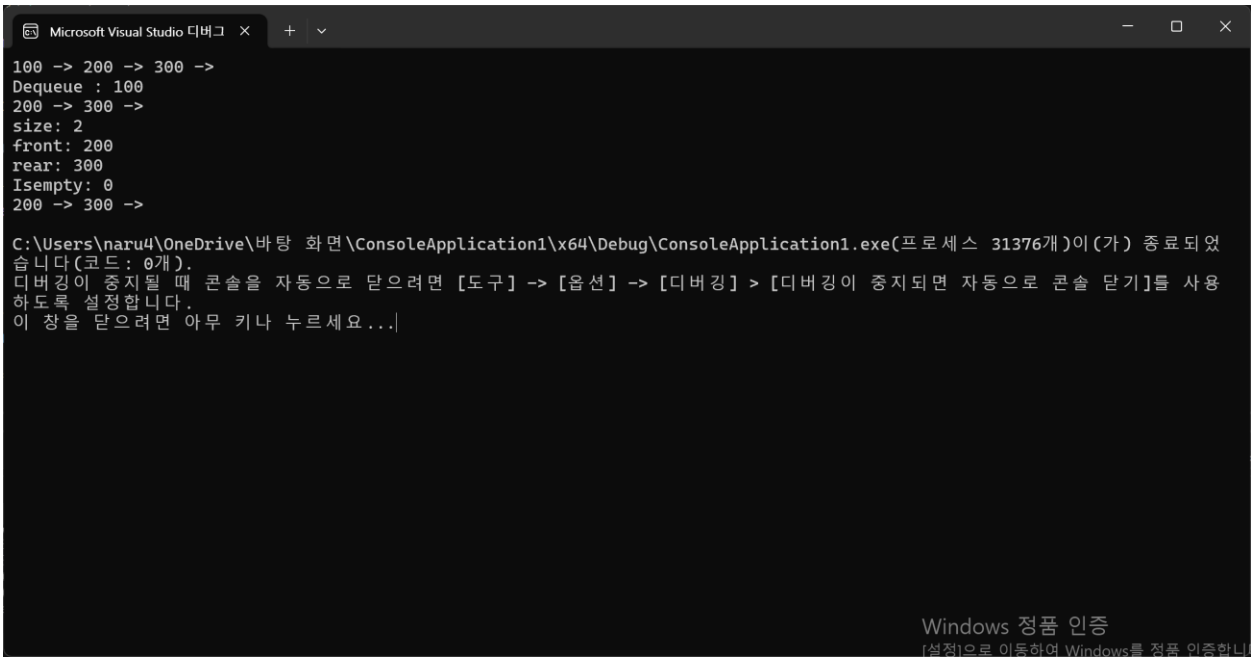
```
#include <stdio.h>
#include <stdlib.h>

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

typedef struct _Queue
{
    Node* front;
    Node* rear;
    int size;
} Queue;

void _Queue(Queue* queue);
```

```
void Enqueue(Queue* queue, int data);
int Dequeue(Queue* queue);
int size(Queue* queue);
int front(Queue* queue);
int rear(Queue* queue);
int isempty(Queue* queue);
void printQueue(Queue* queue);

int main()
{
    Queue queue;
    _Queue(&queue);

    Enqueue(&queue, 100);
    Enqueue(&queue, 200);
    Enqueue(&queue, 300);

    printQueue(&queue);

    printf("Dequeue : %d₩n", Dequeue(&queue));
    printQueue(&queue);

    printf("Size: %d₩n", size(&queue));
    printf("Front: %d₩n", front(&queue));
    printf("Rear: %d₩n", rear(&queue));
    printf("IsEmpty: %d₩n", isempty(&queue));

    printQueue(&queue);
    return 0;
}

void _Queue(Queue* queue)
{
    queue->front = NULL;
    queue->rear = NULL;
    queue->size = 0;
}

void Enqueue(Queue* queue, int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
```

```
    if (queue->rear == NULL)
    {
        queue->front = newNode;
        queue->rear = newNode;
    }
    else
    {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    queue->size++;
}

int Dequeue(Queue* queue)
{
    if (queue->front == NULL)
    {
        printf("큐가 비어있습니다₩n");
        return 0;
    }

    Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;

    if (queue->front == NULL)
    {
        queue->rear = NULL;
    }

    free(temp);
    queue->size--;
    return data;
}

int front(Queue* queue)
{
    if (queue->front == NULL)
    {
        printf("큐가 비어있습니다₩n");
        return 0;
    }
    return queue->front->data;
}
```

```
int rear(Queue* queue)
{
    if (queue->rear == NULL)
    {
        printf("큐가 비어있습니다₩n");
        return 0;
    }
    return queue->rear->data;
}

int size(Queue* queue)
{
    return queue->size;
}

int isempty(Queue* queue)
{
    return (queue->size == 0);
}

void printQueue(Queue* queue)
{
    if (queue->front == NULL)
    {
        printf("큐가 비어있습니다₩n");
        return;
    }
    Node* current = queue->front;
    while (current != NULL)
    {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("₩n");
}
```

실행결과


Microsoft Visual Studio 디버그
100 -> 200 -> 300 ->
Dequeue : 100
200 -> 300 ->
size: 2
front: 200
rear: 300
Isempty: 0
200 -> 300 ->
C:\Users\naru4\OneDrive\바탕 화면\ConsoleApplication1\x64\Debug\ConsoleApplication1.exe(프로세스 31376개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
Windows 정품 인증