데이터구조 Assignment #1

2272027 전자전기공학 심희윤

코드 설명(부분마다 각각 설명을 추가)

```
#include <iostream>

// 2272027 전자전기공학 심희윤

class Node {
public:
    Node* front;
    Node* rear;
    int data;

    Node(int value) : data(value), front(nullptr), rear(nullptr) {}
};
```

- List에서 사용할 **노드 클래스의 기본적인 형태**이다.
- 노드가 가진 data(int)와 list에서 이용할 **[다음 노드를 가르키는 포인터 front]**, **[전 노드를 가르키는 포인터 rear]**가 있다.

```
class List {
private:
    Node* head;
    Node* tail;
    int size;

public:
    List() : head(nullptr), tail(nullptr), size(0) {}
```

- List 클래스의 기본적인 형태이다.
- list의 **맨 앞에 위치한 노드를 가르키는 포인터 head**와 **맨 뒤에 위치한 노드를 가르키는 포인터 tail**이 있으며, **리스트에 있는 노드 개수가 저장되는 size(int)**가 있다.

```
void insertFront(int value) {
    Node* node = new Node(value); //새로운 노드 생성
    node->front = head; // 새로운 노드는 기존의 맨 앞에 있는 노드를
                            front로 가르키게 된다.
    head = node; // head 포인터가 리스트의 맨 앞에 있는 노드를 가르키게
                    한다.

    if(size == 0){ // 리스트가 비어있었을 경우
        tail = node; // 맨 처음 앞으로 들어온 노드를 tail 포인터가 가르키게
                        한다.
    }
    else{ // 리스트가 비어있지 않을 경우
        node->front->rear = node;
    // [새로 들어온 노드의 front가 가르키는 노드의 rear 포인터]는
        [새로 들어온 노드 즉 node]를 가르키게 된다.
    }
    size++; // 리스트 길이 +1
}
```

- **리스트의 앞부분(head쪽)에 새로운 노드를 삽입하는** insertFront 메소드이다.
- 새로운 노드가 삽입되었을 때 일어나는 것
    1. 새 노드가 head가 된다(만약 리스트가 비어있었을 경우에는 새 노드가 tail도 된다.)
    2. 새 노드가 원래 head였던 노드를 front 포인터로 가르키게 된다.
    3. 새 노드 기준 다음 노드의 rear 포인터가 새 노드를 가르키게 된다.
    4. 리스트의 크기가 1 증가한다.

```
void insertBack(int value) {
    Node* node = new Node(value); //새로운 노드 생성
    node->rear = tail; // 새로운 노드는 기존의 맨 뒤에 있는 노드를 rear로
                          가르키게 된다.
    tail = node; // head 포인터가 리스트의 맨 앞에 있는 노드를 가르키게
                    한다.

    if(size == 0){ // 리스트가 비어있었을 경우
        head = node; // 맨 처음 뒤로 들어온 노드를 head 포인터가
                        가르키게 한다.
    }
    else{ // 리스트가 비어있지 않을 경우
        node->rear->front = node;
    // 새로 들어온 노드의 front가 가르키는 노드의
    // rear 포인터는 새로 들어온 노드 즉 node를 가르키게 된다.
    }
    size++; // 리스트 길이 +1
}
```

- **리스트의 뒷부분(tail쪽)에 새로운 노드를 삽입하는** insertBack 메소드이다.
- 새로운 노드가 삽입되었을 때 일어나는 것
    1. 새 노드가 tail이 된다(만약 리스트가 비어있었을 경우에는 새 노드가 head도 된다.)
    2. 새 노드가 원래 tail이었던 노드를 rear 포인터로 가르키게 된다.
    3. 새 노드 기준 전 노드의 front 포인터가 새 노드를 가르키게 된다.
    4. 리스트의 크기가 1 증가한다.

```
    void insert(int index, int value) {
        if(index == 0){ // 맨 앞에 넣는 경우
            insertFront(value);
            return;
        }
        if(index == size){ // 맨 뒤에 넣는 경우
            insertBack(value);
            return;
        }

        if(index < 0 || index > size){ // 에러 처리
            return;
        }

        Node* node = new Node(value); // 삽입할 노드 생성
        Node *n = head;
        for(int i = 1; i < index; i++){// index 만큼 노드를 움직인다.
            n = n->front;
        }

        node->front = n->front; // 삽입할 노드의 front에 n 노드의 front가 가르키는 노드를 할당
        n->front = node; // n 노드의 front가 새로 삽입할 노드를 가르키게 한다.
        node->rear = n; // 삽입할 노드의 rear에 n 노드를 할당
        node->front->rear = node; // 삽입할 노드의 앞에 있는 노드의 rear가 새로 삽입하는 노드를 가르키게 한다.

        size++; // 리스트 길이 +1
    }
```

밑 장에 설명

- **리스트의 어딘가(index로 위치 지정)에 새로운 노드를 삽입하는** insert 메소드.
- 새로운 노드가 삽입되었을 때 일어나는 것
    1. 맨 앞에 넣는 경우(index == 0)와 맨 뒤의 넣는 경우(index == size) 일 때는 각각 insertFront와 insertBack으로 대신 처리가 가능하다.
    2. for문으로 반복해가면서 새 노드를 삽입할 기준이 될 노드를 찾는다. (이때 기준 노드는 새 노드가 삽입됐을 때, 새 노드의 '전 노드'이다)
    3. 기준점이 되는 노드를 찾은 뒤, 기준 노드의 '다음 노드'를 새 노드가 front 포인터로 가르키게 된다. 그리고 기준 노드의 front 포인터는 새 노드를 가르키게 된다.
    4. 새 노드의 rear 포인터는 기준 노드를, 새 노드의 '다음 노드'는 rear 포인터로 새 노드를 가르키게 된다.
    4. 리스트의 크기가 1 증가한다.

```
int erase(int value) {
    for(Node *n = head; n != nullptr ; n = n->front){
            // 맨 앞에서부터 리스트의 끝까지 서치한다.
        if(n->data == value){ // 만약 찾는 값을 가진 노드가 있는 경우
            n->front->rear = n->rear;
            // 해당 노드의 앞에 붙은 노드의 rear 포인터를 수정한다.
            n->rear->front = n->front;
            // 해당 노드의 뒤에 붙은 노드의 front 포인터를 수정한다.
            if(n == head){ // 지운 노드가 헤드였을 경우
                head = n->front;
            // 지운 노드의 front 포인터가 가르키는 노드가 새 head가 된다.
            }
            if(n == tail){ // 지운 노드가 tail였을 경우
                tail = n->rear;
            // 지운 노드의 rear 포인터가 가르키는 노드가 새 tail이 된다.
            }
            size--; // 리스트의 크기 줄이기
            delete n; // 메모리 해제
            return 1; // 지웠다는 뜻의 1 반환
        }
    }
    return 0; // 지울 노드를 찾지 못했다는 뜻의 0 반환
}
```

- 사용자에게서부터 값을 입력받아 그 값을 가진 노드가 리스트 안에 있을 경우, **노드를 삭제하고 1을 반환하며 아닐 경우 0을 반환**하는 erase 메소드.
- erase 메소드가 호출되었을 때 일어나는 것
    1. 현재 노드가 null이 아닐 때까지 값이 맞는 리스트의 노드를 찾아낸다.
    2. 값이 맞는 노드를 찾아냈다면 삭제 노드의 '다음 노드'의 rear 포인터가 삭제 노드의 '전 노드'를 가르키게 한다.
    3. 삭제 노드의 '전 노드'의 front 포인터는 삭제 노드의 '다음 노드'를 가르키게 한다.
    4. 만약 삭제 노드가 head인 경우 삭제 노드의 '다음 노드'가 head가 되고, tail인 경우 삭제 노드의 '전 노드'가 tail이 된다.
    5. 리스트의 크기가 1 감소한다. 노드를 삭제하기 위해 delete를 사용한다.
    6. 삭제 노드를 찾았다면 1, 못 찾았다면 0을 반환한다.

```
int find(int value) {
    for(Node *n = head; n != nullptr ; n = n->front){
    // 맨 앞에서부터 리스트의 끝까지 서치한다.
        if(n->data == value){
            printf("%d is in the list\n", value);
            return 1; // 찾는 노드가 존재한다는 뜻의 1 반환
        }
    }
    printf("%d is not in the list\n", value);
    return 0; // 찾는 노드가 존재하지 않는 뜻의 0 반환
}
```

- 사용자에게서부터 값을 입력받아 그 값을 가진 노드가 리스트 안에 있을 경우, **결과를 출력하고 1을 반환하며 아닐 경우 0을 반환**하는 find 메소드.
- find 메소드가 호출되었을 때 일어나는 것
    1. 현재 노드가 null이 아닐 때까지 값이 맞는 리스트의 노드를 찾아낸다.
    2. 노드를 찾았다면 값 is in the list를 출력하고 1을 반환하고, 못 찾았다면 값 is not in the list를 출력하고 0을 반환한다.

```
    void printList() {
        for(Node *n = head; n != nullptr ; n = n->front){// 맨 앞에서부터
리스트의 끝까지 서치한다.
            printf("%d ", n->data);
        }
        printf("\n");
    }
};
```

- **리스트의 모든 값을 출력하는** printList 메소드.
- printList 메소드가 호출되었을 때 일어나는 것
    1. 현재 노드가 null이 아닐 때까지 모든 노드의 값을 출력한다,

```
int main() {
    List list;

    list.find(1);
    list.insertFront(1);
    list.insertFront(2);
    list.insertFront(3);
    list.printList();

    list.insertBack(10);
    list.insertBack(11);
    list.insertBack(12);
    list.printList();

    list.insert(3, 0);
    list.printList();

    list.find(1);
    list.erase(1);
    list.find(1);

    return 0;
}
```

- 출력과 메소드 호출을 위한 main()