2020105695 김희성

1. 정렬 리스트 자료구조 내의 모든 원소를 역순으로 출력하는 PrintReverse함수를 작성하시오.

순환 리스트의 특성을 고려하여 "PrintReverse()"를 작성할 것.

```
// 이럴바엔 doubly linked list로 구현하는게 좋을 듯
// stack에 저장했다가 한번에 빼는게 시간복잡도상 이득인 듯.
template<class ItemType>
void SortedType<ItemType>::PrintReverse() const
{
    cout << listData->info << endl;
    NodeType<ItemType>* back_iterator = listData;
    while (GetBackNode(back_iterator) != listData)
    {
        back_iterator = GetBackNode(back_iterator);
        cout << back_iterator->info << endl;
    }
    return;
}
```

```
// This function should be called by circular list
template<class ItemType>
NodeType<ItemType>* GetBackNode(const NodeType<ItemType>* const front_node)
{
    NodeType<ItemType>* iterator = const_cast<NodeType<ItemType>*> (front_node);
    while (iterator->next != front_node)
        iterator = iterator->next;
    return iterator;
}
```

2. Linked 기반으로 구현된 Stack내의 모든 원소를 새로운 스택에 복사하 는 Copy함수를 구현하시오.

프로토타입 : void StackType::Copy(StackType& anotherStack)

```
void StackType::Copy(const StackType& anotherStack)
{
    //copy first values
    NodeType* iterator = anotherStack.topPtr;
    this->topPtr = new NodeType;
    this->topPtr->info = iterator->info;

    //copy last values
    NodeType* push_interator = this->topPtr;
    iterator = iterator->next; // 두번째 value 부터
    while (iterator != NULL)
    {
        push_interator->next = new NodeType;
        push_interator = push_interator->next;
        push_interator->info = iterator->info;
        iterator = iterator->next;
    }
    push_interator->next = NULL;
    return;
}
```

3. Double Linked List를 사용하여 다음과 같은 구조를 가지는 텍스트 편 집기를 구현하시오.

a. 위의 구조에 대한 노드 타입을 정의 하시오.

```
struct Line
{
    Line* back;
    Line* next;
    static const int line_size = 81;
    char contents[line_size];
    explicit Line(const char* content,const int line_size);
    explicit Line(const Line& line);
    explicit Line();
};
```

b. 객체를 초기화 하는 클래스 생성자를 구현하세요

```
class TextEditor
{
private:
    Line* const header_padding = new Line;
    Line* const tail_padding = new Line;
    Line* current_line;
    int number_of_line;
```

```
TextEditor::TextEditor(): current_line(header_padding),number_of_line(0)
{
    header_padding->next = tail_padding;
    tail_padding->back = header_padding;
}
```

Dumy Node, padding을 둬서, insert와 delete할 때 code의 복잡성을 줄임. (case 나누지 않도록)

c. GoToTop함수와 GoToBottom를 작성 하시오.

```
void TextEditor::GoToTop()
{
    current_line = header_padding->next;
}

void TextEditor::GoToBottom()
{
    current_line = tail_padding->back;
}
```

d. c의 Big-O를 생각해보고, 위 함수가 O(1)이 되는 경우를 서술 하시오.

Front를 가르키는 포인터와 rear(tail)을 가르키는 노드가 있으면, O(1)로 한번에 접근 가능하다.

e. InsertItem함수를 작성 하시오.

```
void TextEditor::InsertItem(const char newline[])
{
    Line* added_line = new Line(newline, strlen(newline));
    strcpy_s(added_line->contents, strlen(newline)+1, newline);
    added_line->back = current_line;
    added_line->next = current_line->next;
    current_line->next->back = added_line;
    current_line->next = added_line;
    current_line = added_line;
}
```

Dumy Node padding을 사용하여, 코드의 복잡도를 낮춤.