

CSED 232 Object-Oriented Programming (Spring 2024)

Programming Assignment #6

Due date : 6월 7일 23시 59분 59초

담당 조교 : 신제현 (tlswpgus22@postech.ac.kr)

[안내사항]

1. 모든 문제는 C++의 standard 입출력(i.e. cin, cout)을 기본으로 합니다.
2. 채점용 testcase를 통과한 개수에 비례하여 프로그램 기능 점수가 부여됩니다.
 - (통과 case / 총 case) * 채점 기준에 명시된 프로그램 기능 점수
 - Testcase는 공개되지 않습니다.
3. main.cpp와 test.cpp에 있는 예시코드를 활용하여 문제들의 입출력 예시와 동일하게 나오는지 확인할 수 있습니다.

[감점]

1. 제출 기한이 지나면 얻은 총점의 20% 감점
2. 하루(24시간) 늦을 때마다 추가 20%씩 감점
 - 1일 이내 : 20% 감점, 2일 이내 : 40% 감점, 3일 이내 : 60% 감점, 4일 이내 : 80% 감점
 - 4일 이상 지연 : 0점
3. 컴파일이 정상적으로 되지 않을 경우 프로그램 기능 점수 0점

[제출방식]

채점은 Windows Visual Studio 2022 환경에서 이루어집니다. 파일을 업로드하실 때, **개발환경 파일**의 “파일 제출” 페이지에 써 있는 대로 맞춰 올려 주시기 바랍니다. 폴더명은 문제#_학번(e.g., **prob6_20230000**)으로 만들어 주십시오. 또한 문제 폴더 안에 각 문제에 해당하는 **Report(prob6_20230000_report.pdf)**도 같이 넣어서 zip파일로 만든 후 제출해 주시기 바랍니다.

이 때, 첨부파일에 포함된 **.h, .cpp 파일만을 이용하여** 제출해주시기 바랍니다. **MultiHeadList.h**와 **Syllogism.cpp**만을 작성하여 제출하시면 됩니다. 제출은 반드시 PLMS를 통해 제출해주시기 바랍니다. 이메일 제출은 인정되지 않습니다. 4일 이상 지연 제출할 경우 0점이므로 4일(6월 11일 23시 59분 59초)이 지난 이후는 PLMS를 통해 제출하실 수 없습니다. 양식을 지키지 않을 시 감점

이 존재합니다.

제출파일 예시) prob6_20230000.zip (prob6_20230000 [폴더], prob6_20230000_report.pdf)

[채점기준]

1. 프로그램 기능(설계 및 구현) - 85%

- 프로그램이 요구 사항을 모두 만족하면서 올바르게 실행되는가?
- 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
- 각 문제에서 제시한 세부 조건의 유의사항을 모두 만족하였는가?
- 입력과 출력이 주어진 형식에 맞게 프로그램이 잘 작동하는가?
- MultiHead 클래스 65점
- Syllogism 클래스 20점

2. 프로그램 가독성 - 5%

- 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
- 변수 명이 무엇을 의미하는지 파악하기 쉬운가?
- 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

3. 보고서 구성 및 내용, 양식 - 10%

- 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
- 보고서의 양식을 잘 따랐는가?
- 각 문제에서 제시한 질문이 있다면, 그에 대한 답변이 충분한가?

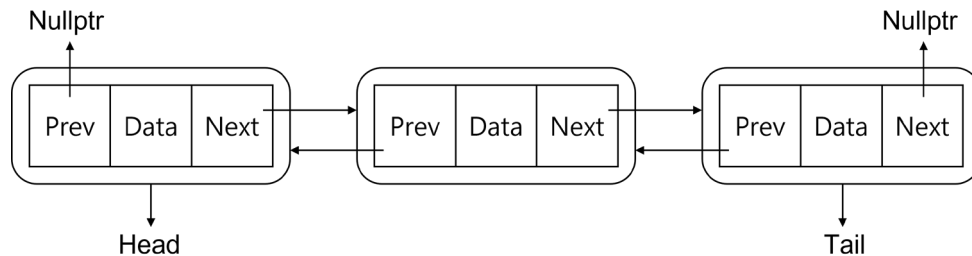
[주의사항]

다른 사람의 프로그램이나 인터넷에 있는 프로그램을 단순히 복사(copy)하거나 수정해서 제출하면 부정행위로 간주됩니다. 부정행위 발견 시 'F' 학점을 받을 수 있으며, 학과에서 정한 기준에 따라 추가적인 불이익이 있을 수 있습니다.

문제 1. MultiHeadList (배점 65점)

Head가 여러 개 있는 Doubly Linked List를 template을 통해 구현하고자 한다.

[DoublyLinkedList]

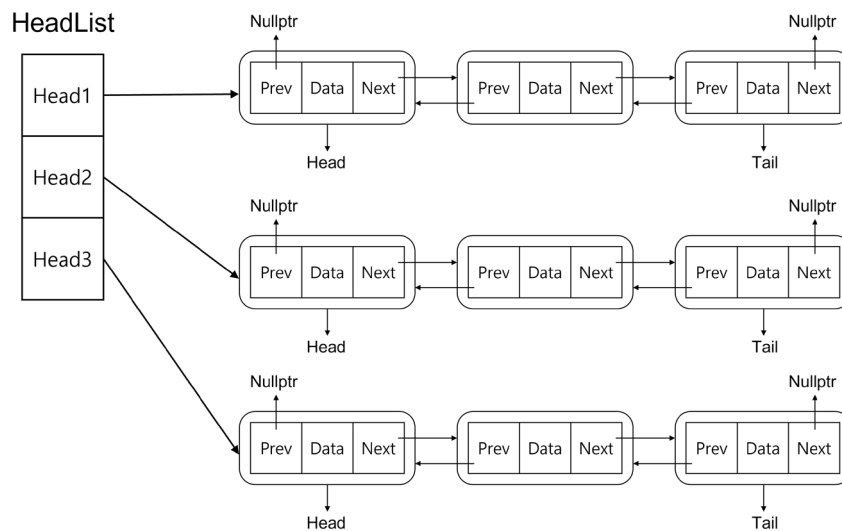


노드(둥근네모)는 이전 노드를 가리키는 Prev(Node *)와 정보를 담고 있는 Data(T), 다음 노드를 가리키는 Next(Node *)로 구성된다. 제일 처음의 노드를 Head라고 부르며, 제일 마지막의 노드는 Tail이라 불린다. Head의 Prev와 Tail의 Next는 nullptr를 가리킨다.

[MultiHeadList]

MultiHeadList는 Head가 여러 개인 doubly linked list이다.

Head의 주소는 headList(std::vector<Node<T>*>)에서 관리된다.



[변수 및 함수]

MultiHeadList는 다음과 같은 변수를 가진다.

표 1 MultiHeadList class의 변수

변수	타입	설명
headList	std::vector<Node<T>*>	Head의 주소가 담겨있는 vector

MultiHeadList는 다음과 같은 함수를 가진다. 필요시 기타 함수를 작성하는 것은 가능하며 추가적으로 함수를 구현하는 것 역시 가능하다.

표 2 MultiHeadList class의 함수

함수	설명
void push_back(const T& data, int headIdx=-1)	1. headIdx번째 List 맨 끝에 data 추가 2. headIdx가 음수이거나 headList에 저장되어 있는 head의 인덱스를 초과하는 값이 들어올 때는 새로운 head 생성하여 추가
void push_front(const T& data, int headIdx=-1)	1. headIdx번째 List 맨 끝에 data 추가 2. headIdx가 음수이거나 headList에 저장되어 있는 head의 인덱스를 초과하는 값이 들어올 때는 새로운 head 생성하여 추가
void insert(Iterator pos, const T& data)	pos가 가리키는 곳 이전에 data 삽입
void pop_back(int headIdx)	headIdx번째 List 맨 끝 data 삭제
void pop_front(int headIdx)	headIdx번째 List 맨 앞 data 삭제
void merge(int frontHeadIdx, int backHeadIdx)	frontHeadIdx번째 List 뒤에 backHeadIdx번째 List 붙이기
bool erase(const T& data, int targetHeadIdx)	1. targetHeadIdx번째 List에 data가 있다면 삭제 2. head부터 시작하여 제일 처음 발견된 data만 삭제
bool erase(Iterator pos)	pos가 가리키는 곳 data 삭제
int headSize()	headList의 크기를 반환
Iterator begin(int headIdx)	headIdx번째 List의 begin()
Iterator end()	Nullptr로 구현
Reverseliterator rbegin(int headIdx)	1. headIdx번째 List의 rbegin() 2. headIdx번째 List의 tail부터 역순으로 움직인다
Reverseliterator rend()	Nullptr로 구현

[Iterator]

MultiheadList 내부에서 정의되는 class로 Iterator와 Reverseliterator를 모두 구현해야한다. Iterator는 각 head별로 구현이 될 수 있도록 한다. 표2에서 볼 수 있듯이 multiheadlist의 begin(i)은 i번째 list의 head를 가리키는 iterator가 된다.

[변수 및 함수]

Iterator와 Reverseliterator는 다음과 같은 변수를 가진다.

표 3 Iterator와 Reverseliterator class의 변수

변수	타입	설명
curr	Node<T>*	List에서 현재 노드를 가리키는 포인터

Iterator와 Reverseliterator는 다음과 같은 함수를 가진다.

표 4 Iterator와 Reverseliterator class의 함수

함수	설명
Iterator/Reverseliterator (Node<T>* node)	생성자
operator++	전위/후위 연산자 구현 (Iterator와 Reverseliterator의 동작 반대)
operator--	전위/후위 연산자 구현 (Iterator와 Reverseliterator의 동작 반대)
operator+(int n)	n번만큼 ++ 연산
operator-(int n)	n번만큼 -- 연산
operator!=	!=
operator==	==
operator*()	curr의 data를 출력

[입출력 예시]

MultiHeadList<int> mhList 을 선언한 이후, 다음과 같은 결과를 확인할 수 있다.

```

push_back(1)
0 : 1
=====
push_back(2, 0)
0 : 1 2
=====
push_back(3, 1)
0 : 1 2
1 : 3
=====
push_back(4)
0 : 1 2
1 : 3
2 : 4
=====
push_back(5,1)
0 : 1 2
1 : 3 5
2 : 4
=====
push_front(6, 2)
0 : 1 2
1 : 3 5
2 : 6 4
=====
pop_back(2)
0 : 1 2
1 : 3 5
2 : 6
=====
pop_front(2)
0 : 1 2
1 : 3 5
=====
push_front(7)
0 : 1 2
1 : 3 5
2 : 7
=====
merge(2,1)
0 : 1 2
1 : 7 3 5
=====
erase(7,0)
0 : 1 2
1 : 7 3 5
=====
erase(7,1)
0 : 1 2
1 : 3 5
=====
insert(mhList.begin(1), 8)
0 : 1 2
1 : 8 3 5
=====
insert(++mhList.begin(1), 9)
0 : 1 2
1 : 8 9 3 5
=====
erase(mhList.begin(1)+2)
0 : 1 2
1 : 8 9
2 : 5
=====

```

문제 2. n 단 논법 (배점 20 점)

문제1에서 구현한 MultiHeadList를 이용하여 n단논법을 구현해보고자 한다.

“A 이면 B이다” 라는 논리는 `std::pair<std::string, std::string>`에 저장된다. 논리가 연결되지 않는 경우에는 새로운 head를 만들어서 headList에 추가한다. 모든 논리를 취합한 뒤, 질문을 던졌을 때 최종 결론을 내는 것이 문제의 목표이다. 단, 순환논리는 발생하지 않는다고 가정한다. 다음의 예시를 살펴보자. 번호 순서대로 n단논법을 MultiHeadList에 저장했을 때 결과는 다음과 같다.

1. A이면 B이다. `0 : A->B`
2. B이면 C이다. `0 : A->B->C`
3. E이면 F이다. `0 : A->B->C`
`1 : E->F`
4. D이면 E이다. `0 : A->B->C`
`1 : D->E->F`
5. C이면 D이다. `0 : A->B->C->D->E->F`

2 번까지 논리를 저장한 후, “B 이면?”이라고 질문했을 때 정답은 “C 이다” 이다.

5 번까지 논리를 저장한 후, “B 이면?”이라고 질문했을 때 정답은 “F 이다” 이다.

[변수 및 함수]

Syllogism class는 다음과 같은 변수를 가진다.

표 5 Syllogism class의 변수

변수	타입	설명
syl	MultiHeadList<std::pair<std::string, std::string>>	논리가 저장되는 MultiHeadList

Syllogism class는 다음과 같은 함수를 가진다. 기능을 위해 필요한 다른 함수 사용 가능하다.

표 6 Syllogism class의 함수

함수	설명
<code>void put(const std::pair<std::string, std::string>& argument)</code>	논법을 넣고 해당 논법이 잘 연결될 수 있도록 구성한다. Pair의 첫번째 값(first)가 “~이면”에 해당하며, pair의 두번째 값(second)가 “~이다.”에 해당한다.
<code>void qna(const std::string& q)</code>	질문 q를 던졌을 때 (“q이면”) n단 논법을 거친 후, 최종 결론을 대답한다.
<code>void print()</code>	MultiHeadList의 전체적인 상황을 표현하는 함수로 이미 구현이 되어 있다.

[입출력 예시]

입력	출력
Syllogism syl;	=====
syl.put(make_pair("A", "B"));	0 : A->B
syl.print();	=====
syl.put(make_pair("B", "C"));	0 : A->B->C
syl.print();	=====
syl.put(make_pair("E", "F"));	0 : A->B->C
syl.print();	1 : E->F
syl.put(make_pair("D", "E"));	=====
syl.print();	0 : A->B->C
syl.qna("B");	1 : D->E->F
syl.put(make_pair("C", "D"));	If B, then C
syl.print();	=====
syl.qna("B");	0 : A->B->C->D->E->F
	If B, then F