2024 Spring OOP Assignment Report

과제 번호 : 6
학번 : 20230024
이름 : 문요준
Povis ID : yojun313

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. 프로그램 개요

1-1. 개요

이 프로그램은 Head가 여러 개 있는 Doubly Linked List를 구현하고 이를 바탕으로 n단 논법을 구현하는 것이다.

1-2. 프로그램 실행 방법

Visual Studio와 같은 IDE에서 모든 파일을 포함하는 프로젝트를 생성하고 각 파일을 컴파일하고 링크, 컴파일된 실행파일을 실행하면 main()함수가 실행되고 구현한 클래스에 대한 테스트가 자동으로 이루어지게 된다. main.cpp는 Syllogism 객체를 생성하고 여러 개의 논법을 추가한 후 질문에 대한 답을 출력하고 test.cpp는 MultriHeadLIst의 다양한 기능을 테스트하고 결과를 출력한다.

1-3. 프로그램 파일 설명

1. main.cpp

- 설명: 프로그램의 진입점(main 함수)이 포함된 파일로, Syllogism 클래스와 Test 클래스를 사용하여 논법을 추가하고, 질문에 대한 답변을 출력하며, MultiHeadList의 기능을 테스트하는 파일이다
- 주요 기능:
    - Syllogism 객체를 생성하고, 여러 논법을 추가하고 출력한다
    - 질문에 대한 답변을 출력한다
    - Test 객체를 생성하여 MultiHeadList의 기능을 테스트하고 출력한다

2. test.cpp

- 설명: MultiHeadList 클래스의 다양한 기능을 테스트하는 파일로, 리스트의 추가, 삭제, 병합 등을 수행하는 파일이다
- 주요 기능:
    - MultiHeadList의 push_back, push_front, pop_back, pop_front, merge, erase, insert 등의 기능을 테스트하고 결과를 출력한다
    - print 함수를 통해 리스트의 현재 상태를 출력한다

3. syllogism.cpp

- 설명: Syllogism 클래스를 구현한 파일로, 논법을 추가하고 질문에 답변하는 기능을 제공하는 파일이다
- 주요 기능:
    - 논법을 추가하는 put 메서드
    - 질문에 답변하는 qna 메서드
    - 현재 저장된 논법들을 출력하는 print 메서드

4. MultiHeadList.h

- 설명: MultiHeadList 클래스를 정의한 헤더 파일로, 다중 헤드를 가지는 리스트 구조를 구현한 파일이다
- 주요 기능:
    - 노드 구조를 정의하고, 리스트에 노드를 추가, 삭제, 병합하는 메서드들을 제공한다

2. 프로그램의 구조 및 알고리즘

2-1. Class 설명

1. Class: MultiHeadList

멤버 변수 (Member Variables)

- headList:
    - 타입: vector<Node<T>*>
    - 설명: 리스트의 각 헤드를 가리키는 포인터들의 벡터다. 각 요소는 리스트의 헤드(첫 번째 노드)를 가리키는 포인터다.

Node 구조체

- data:
    - 타입: T
    - 설명: 노드가 저장하는 데이터다.
- prev:
    - 타입: Node<T>*
    - 설명: 이전 노드를 가리키는 포인터다.
- next:
    - 타입: Node<T>*
    - 설명: 다음 노드를 가리키는 포인터다.

Iterator 클래스

- curr:
    - 타입: Node<T>*
    - 설명: 현재 노드를 가리키는 포인터다.

ReverseIterator 클래스

- curr:
    - 타입: Node<T>*
    - 설명: 현재 노드를 가리키는 포인터다.

멤버 함수 (Member Functions)

생성자

- MultiHeadList():
    - 설명: MultiHeadList 클래스의 기본 생성자다. 특별한 초기화는 필요 없다.

1. headSize()

- int headSize() const:
    - 설명: headList의 크기를 반환한다.
    - 내부 동작: headList.size()를 호출하여 벡터의 크기를 반환한다.

2. push_back()

- void push_back(const T& data, int headIdx = -1):
    - 설명: 지정된 인덱스의 리스트 끝에 데이터를 추가하거나, 인덱스가 없으면 새로운 리스트를 생성한다.
    - 내부 동작:

1. headIdx가 유효한 범위에 있는지 확인한다.
2. 유효하지 않으면 새로운 노드를 생성하여 headList에 추가한다.
3. 유효하면 지정된 인덱스의 리스트 끝까지 이동한 후 새 노드를 추가한다.

3. push_front()

- void push_front(const T& data, int headIdx = -1):
  - 설명: 지정된 인덱스의 리스트 앞에 데이터를 추가하거나, 인덱스가 없으면 새로운 리스트를 생성한다.
  - 내부 동작:
    1. headIdx가 유효한 범위에 있는지 확인한다.
    2. 유효하지 않으면 새로운 노드를 생성하여 headList에 추가한다.
    3. 유효하면 지정된 인덱스의 리스트 앞에 새 노드를 추가한다.

4. insert()

- void insert(Iterator pos, const T& data):
  - 설명: 지정된 위치의 앞에 데이터를 삽입한다.
  - 내부 동작:
    1. pos가 가리키는 노드를 가져온다.
    2. 새로운 노드를 생성하고, pos의 노드 앞에 삽입한다.
    3. 이전 노드와 다음 노드의 포인터를 업데이트한다.

5. pop_back()

- void pop_back(int headIdx):
  - 설명: 지정된 인덱스의 리스트 끝에서 데이터를 제거한다.
  - 내부 동작:
    1. headIdx가 유효한 범위에 있는지 확인한다.
    2. 유효하면 리스트 끝까지 이동하여 마지막 노드를 삭제한다.
    3. 삭제 후 리스트가 비어있으면 headList에서 해당 리스트를 제거한다.

6. pop_front()

- void pop_front(int headIdx):
  - 설명: 지정된 인덱스의 리스트 앞에서 데이터를 제거한다.
  - 내부 동작:
    1. headIdx가 유효한 범위에 있는지 확인한다.
    2. 유효하면 리스트 앞의 첫 번째 노드를 삭제한다.
    3. 삭제 후 리스트가 비어있으면 headList에서 해당 리스트를 제거한다.

7. merge()

- void merge(int frontHeadIdx, int backHeadIdx):
  - 설명: 두 개의 리스트를 병합한다.
  - 내부 동작:

1. frontHeadIdx와 backHeadIdx가 유효한 범위에 있는지 확인한다.
2. frontHeadIdx의 리스트 끝으로 이동하여 backHeadIdx의 리스트를 연결한다.
3. backHeadIdx의 헤드를 삭제하고 headList에서 제거한다.

8. erase(const T& data, int targetHeadIdx)

- bool erase(const T& data, int targetHeadIdx):
    - 설명: 지정된 인덱스의 리스트에서 특정 데이터를 삭제한다.
    - 내부 동작:
        1. targetHeadIdx가 유효한 범위에 있는지 확인한다.
        2. 유효하면 리스트를 순회하며 데이터를 찾는다.
        3. 데이터를 찾으면 해당 노드를 삭제하고 이전 및 다음 노드의 포인터를 업데이트한다.
        4. 삭제 후 리스트가 비어있으면 headList에서 해당 리스트를 제거한다.

9. erase (Iterator)

- bool erase(Iterator pos):
    - 설명: 지정된 위치의 노드를 삭제한다.
    - 내부 동작:
        1. pos가 가리키는 노드를 가져온다.
        2. 이전 및 다음 노드의 포인터를 업데이트하여 현재 노드를 리스트에서 제거한다.
        3. 필요하면 다음 노드를 새로운 헤드로 추가한다.
        4. headList에서 빈 리스트를 제거한다.

10. rremove()

- bool rremove(Iterator pos):
    - 설명: 지정된 위치의 노드를 삭제하는 또 다른 방법이다. Syllogism.cpp에서 쓰인다.
    - 내부 동작: erase(Iterator) 메서드와 동일하다.

11. begin()

- Iterator begin(int headIdx):
    - 설명: 지정된 인덱스의 리스트 시작을 가리키는 반복자를 반환한다.
    - 내부 동작: headList[headIdx]를 가리키는 반복자를 생성하여 반환한다.

12. end()

- Iterator end():
    - 설명: 리스트 끝을 가리키는 반복자를 반환한다.
    - 내부 동작: nullptr을 가리키는 반복자를 반환한다.

13. rbegin()

- ReverseIterator rbegin(int headIdx):

- 설명: 지정된 인덱스의 리스트 끝을 가리키는 역방향 반복자를 반환한다.
- 내부 동작: 리스트 끝을 찾기 위해 순회한 후 해당 노드를 가리키는 역방향 반복자를 반환한다.

14. rend()

- ReverseIterator rend():
  - 설명: 리스트의 역방향 끝을 가리키는 반복자를 반환한다.
  - 내부 동작: nullptr을 가리키는 역방향 반복자를 반환한다.

15. findIndex()

- int findIndex(Iterator pos) const:
  - 설명: 지정된 반복자가 가리키는 노드의 인덱스를 찾는다.
  - 내부 동작:
    1. pos가 가리키는 노드를 가져온다.
    2. 각 리스트를 순회하며 노드를 찾는다.
    3. 노드를 찾으면 해당 인덱스를 반환한다.
    4. 찾지 못하면 -1을 반환한다.

2. Class: Syllogism

멤버 변수(Member Variables)

- syl:
  - 타입: MultiHeadList<pair<string, string>>
  - 설명: 논법을 저장하고 관리하기 위한 MultiHeadList 객체다. 각 노드는 논법의 전제와 결론을 저장하는 pair<string, string> 타입이다.

멤버 함수(Member Functions)

1. stringEquals()

- bool stringEquals(const string& str1, const string& str2):
  - 설명: 두 문자열을 비교하는 헬퍼 함수다.
  - 내부 동작: 두 문자열이 동일한지 비교하고, 동일하면 true, 그렇지 않으면 false를 반환한다.

2. put()

- void put(const pair<string, string>& argument):
  - 설명: 새로운 논법을 추가하는 함수다.
  - 내부 동작:

1. argument의 첫 번째 요소가 기존 논법의 두 번째 요소와 일치하는지 확인한다.
2. 일치하면 해당 논법 리스트의 끝에 argument를 추가한다.
3. 추가 후 연결할 수 있는 노드가 있으면 연결하고, 연결된 노드는 삭제한다.
4. argument의 두 번째 요소가 기존 논법의 첫 번째 요소와 일치하는지 확인한다.
5. 일치하면 해당 논법 리스트의 앞에 argument를 추가한다.
6. 추가 후 연결할 수 있는 노드가 있으면 연결하고, 연결된 노드는 삭제한다.
7. 기존 논법과 연결되지 않으면 새로운 리스트 헤드를 생성하여 argument를 추가한다.

3. qna()

- void qna(const string& q):
  - 설명: 주어진 질문에 대해 저장된 논법을 바탕으로 답변을 제공하는 함수다.
  - 내부 동작:
    1. 주어진 질문을 바탕으로 논법 리스트에서 첫 번째 요소가 질문과 일치하는 노드를 찾는다.
    2. 일치하는 노드를 찾으면 해당 논법의 두 번째 요소를 결과로 저장한다.
    3. 저장된 결과를 바탕으로 연결된 논법을 계속해서 찾아 최종 결과를 도출한다.
    4. 결과를 출력한다.
    5. 일치하는 논법을 찾지 못하면 "No conclusion can be drawn from" 메시지를 출력한다.

4. print()

- void print():
  - 설명: 현재 저장된 논법 리스트의 상태를 출력하는 함수다.
  - 내부 동작:
    1. syl 리스트의 각 헤드를 순회한다.
    2. 각 헤드에 연결된 논법을 출력 형식에 맞게 출력한다.
    3. 각 리스트는 인덱스로 구분하여 출력된다.

2-2. 전체적 알고리즘 설명 – main.cpp

1. 프로그램 시작

- main.cpp 파일의 main 함수에서 프로그램이 시작된다
- Syllogism 객체를 생성한다

- 여러 논법을 추가하고 출력하며, 질문에 대한 답변을 출력한다
- Test 객체를 생성하여 MultiHeadList의 기능을 테스트한다

2. Syllogism 객체의 역할
   - 논법을 저장하고 관리하는 역할을 한다
   - put 메서드를 사용하여 새로운 논법을 추가한다
   - qna 메서드를 사용하여 주어진 질문에 대한 답변을 제공한다
   - print 메서드를 사용하여 현재 저장된 논법 리스트의 상태를 출력한다

3. put 메서드의 동작
   - put 메서드는 주어진 논법을 리스트에 추가하는 역할을 한다
   - 주어진 논법의 첫 번째 요소가 기존 리스트의 두 번째 요소와 일치하는지 확인한다
   - 일치하면 해당 리스트의 끝에 논법을 추가하고, 연결할 수 있는 노드가 있으면 연결한 후 삭제한다
   - 주어진 논법의 두 번째 요소가 기존 리스트의 첫 번째 요소와 일치하는지 확인한다
   - 일치하면 해당 리스트의 앞에 논법을 추가하고, 연결할 수 있는 노드가 있으면 연결한 후 삭제한다
   - 기존 논법과 연결되지 않으면 새로운 리스트 헤드를 생성하여 논법을 추가한다

4. qna 메서드의 동작
   - qna 메서드는 주어진 질문에 대해 저장된 논법을 바탕으로 답변을 제공한다
   - 주어진 질문을 바탕으로 논법 리스트에서 첫 번째 요소가 질문과 일치하는 노드를 찾는다
   - 일치하는 노드를 찾으면 해당 논법의 두 번째 요소를 결과로 저장한다
   - 저장된 결과를 바탕으로 연결된 논법을 계속해서 찾아 최종 결과를 도출한다
   - 결과를 출력한다
   - 일치하는 논법을 찾지 못하면 "No conclusion can be drawn from" 메시지를 출력한다

5. print 메서드의 동작
   - print 메서드는 현재 저장된 논법 리스트의 상태를 출력한다
   - 리스트의 각 헤드를 순회하며 각 헤드에 연결된 논법을 출력 형식에 맞게 출력한다
   - 각 리스트는 인덱스로 구분하여 출력된다

6. Test 클래스의 역할
   - MultiHeadList의 다양한 기능을 테스트한다
   - 리스트에 데이터를 추가, 삭제, 병합하는 기능을 테스트하고 결과를 출력한다

3. 토론 및 개선

3-1. 배우거나 깨달은 내용

- 다중 헤드 리스트 구조:
    - 다중 헤드를 가지는 리스트 구조를 이해하고, 이를 구현하는 방법을 배울 수 있었다.
    - 각 리스트가 독립적으로 존재하면서도, 필요에 따라 병합하거나 나눌 수 있는 구조를 이해할 수 있었다.
- 반복자(iterator)의 사용:
    - 반복자를 사용하여 리스트를 순회하는 방법을 배울 수 있었다.
    - 정방향 및 역방향 반복자를 구현하고 사용하는 방법을 이해할 수 있었다

3-2. 개선 방향

클래스 분리: Syllogism 클래스와 MultiHeadList 클래스를 별도의 파일로 분리하여, 코드의 가독성과 유지보수성을 높일 수 있다.