게임알고리즘기초과정

과 목 명 : 게임기초프로그래밍

능력단위 : C/C++ 프로그래밍

제출일자 : 2023년 07월 04일

포트폴리오 : 파일 입출력 및 템플릿을 적용한 학생관리 프로그램 제작

작성자 : 오시진

<제출내역>

1. 학생 관리 프로그램 구현 프로젝트
2. 학생 관리 프로그램 구현 분석 및 세부 문서

<CONTENTS>

1. 프로젝트 소개 및 개요
   1. 프로젝트 소개
   2. 프로젝트 주요 기술
2. 프로젝트 설계 및 다이어그램
   1. 시퀀스 다이어그램
   2. 클래스 다이어그램
      1. 프로젝트에서 사용된 변수 역할
      2. 프로젝트에서 사용된 함수 역할
3. 최종 결과 및 추가 내용
   1. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
      1. 현재 프로젝트 설계(구현 내용)
      2. 향후 개발 내용
   2. 최종 결과

1. 프로젝트 소개 및 개요
   1. 프로젝트 소개

대상 프로젝트는 객체지향프로그래밍 방식으로 학생관리프로그램을 제작한다. 자료구조 중 양방향 연결리스트를 직접 구현하여 학생 정보를 관리한다. 연결리스트의 각 노드들은 포인터로 연결되어 있고 데이터 또한 포인터로 관리하기 때문에 동적 할당 등 메모리 관리 측면에서 프로그램을 설계한다. 또한 확장성과 재사용성을 위해 템플릿을 활용하여 연결리스트 등을 모듈화한다. 그리고 파일에 정보를 저장하고 불러오는 기능을 학습한다.

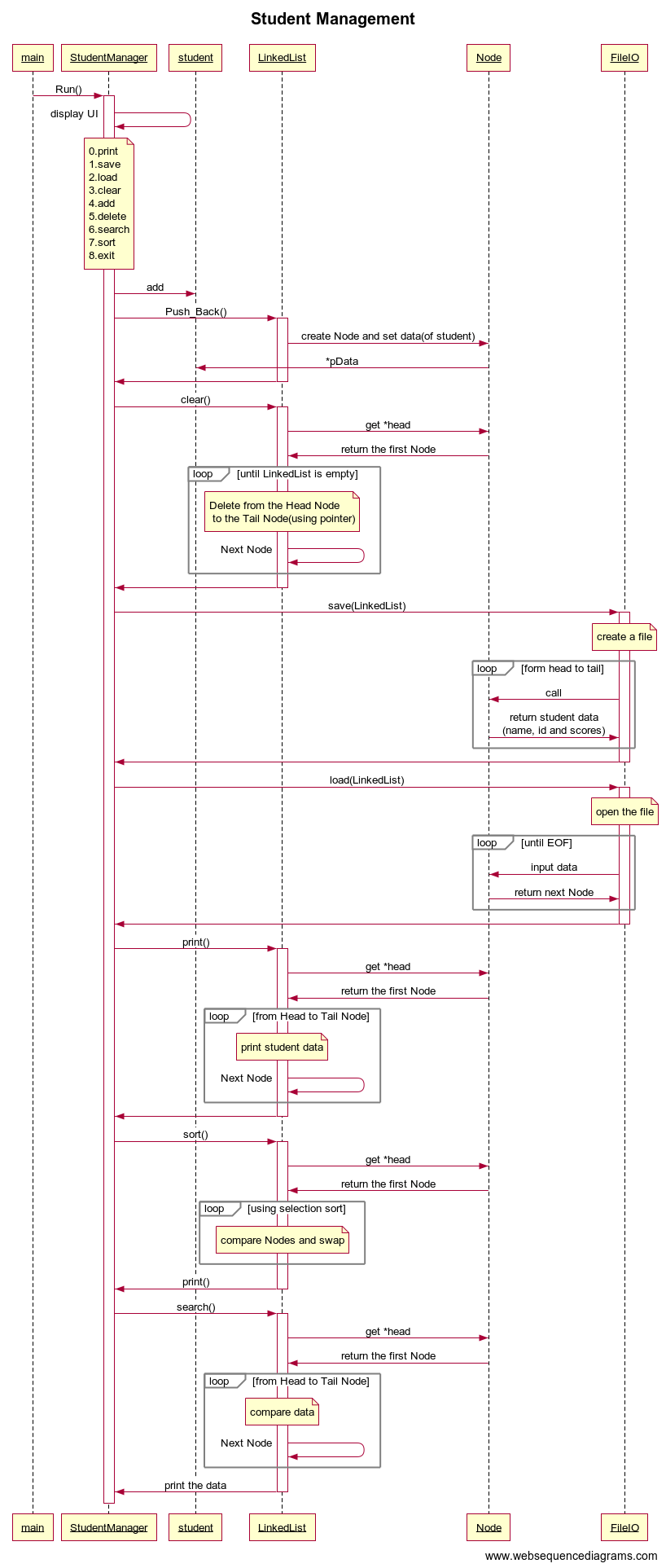
* 1. 프로젝트 주요 기술
     1. 템플릿 연결리스트

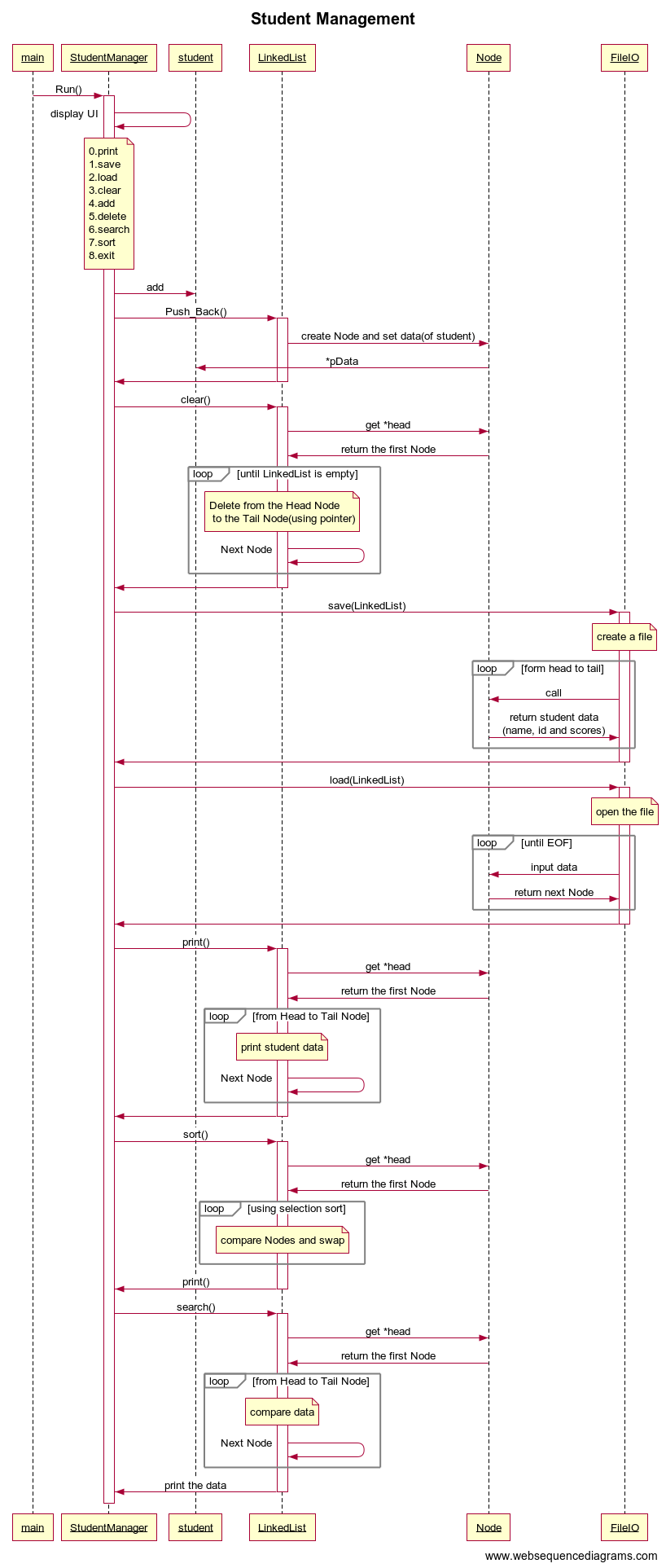
템플릿을 사용하여 특정 데이터 타입에 종속되지 않고 여러 타입에 대해 동작하는 연결리스트를 작성하였다. 이렇게 구축된 연결리스트는 높은 범용성으로 다른 프로젝트에서 필요할 경우 코드의 재사용이 가능하다. 연결리스트는 양방향으로 연결되어 있으며 Head와 Tail 포인터를 사용하여 시작과 끝 노드를 관리한다. 연결리스트의 특성 상 검색 기능은 O(n)의 성능을 가질 수밖에 없지만, 정렬 기능은 선택 정렬 알고리즘을 적용하였다. 사용자는 인터페이스를 통하여 데이터의 검색, 삭제, 추가, 정렬 등의 기능을 이용할 수 있다.

* + 1. 파일 입출력

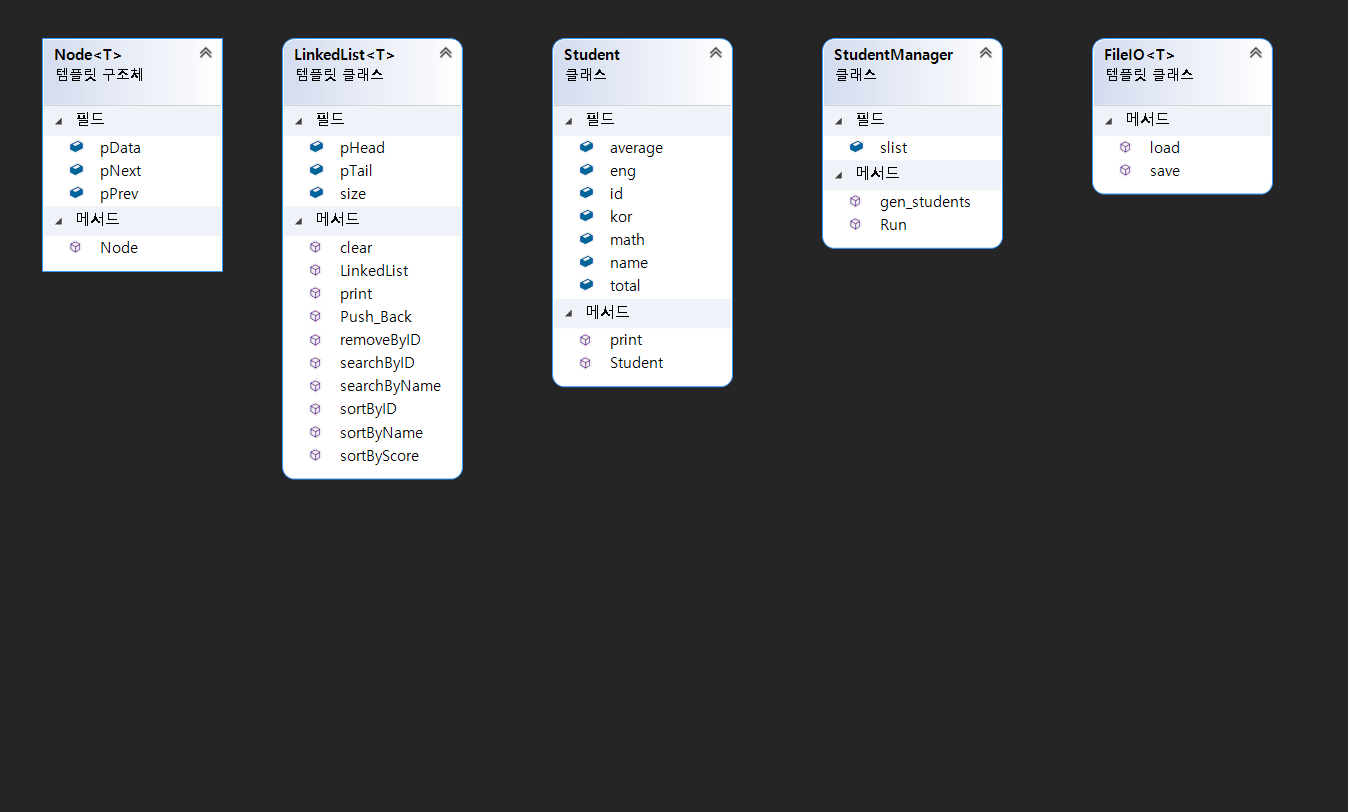
해당 프로젝트에서는 텍스트 파일을 이용하여 데이터를 저장하고 불러온다. 따라서 프로그램을 동작하지 않고도 학생 데이터를 열람할 수 있으며 필요에 따라 텍스트파일에 직접 접근하여 수정하거나 삭제하는 것 또한 가능하다. 이는 보안 상 덜 안전하기 때문에 사용자의 주의가 필요한 부분이다.

1. 프로젝트 설계 및 다이어그램
   1. 시퀀스 다이어그램





* 1. 클래스 다이어그램



* + 1. 프로젝트에서 사용된 변수 역할

|  |  |  |
| --- | --- | --- |
| 클래스명 | 변수 | 내용 |
| Node | T\* pData | 노드에 저장할 데이터 멤버 변수 |
| Node<T>\* pNext | 다음 노드를 가리키는 노드 포인터 변수 |
| Node<T>\* pPrev | 이전 노드를 가리키는 노드 포인터 변수 |
| LinkedList | Node<T>\* pHead | 연결리스트의 첫 번째 노드를 가리키는 노드 포인터 변수 |
| Node<T>\* pTail | 연결리스트의 마지막 노드를 가리키는 노드 포인터 변수 |
| size | 연결리스트에 저장된 노드의 개수 |
| Student | id | 학생 번호 |
| name | 학생 이름 |
| kor | 국어 점수 |
| eng | 영어 점수 |
| math | 수학 점수 |
| total | 총 점수 |
| average | 평균 점수 |
| Student  Manager | slist | 학생 데이터가 저장될 연결리스트 |

* + 1. 프로젝트에서 사용된 함수 역할

|  |  |  |
| --- | --- | --- |
| 클래스명 | 함수 | 내용 |
| Node | Node(T\* t) | 노드를 초기화하고 데이터를 저장 |
| LinkedList | Void Push\_Back(T\* s) | 새로운 노드 생성과 연결 |
| Bool removeByID(int id) | 해당 id의 노드를 삭제 |
| Void clear() | 모든 노드를 삭제 |
| Void print() | 연결리스트의 모든 데이터를 출력 |
| T\* searchByName(const string& name) | 이름으로 검색 |
| T\* searchByID(const int& id) | ID로 검색 |
| Void sortByName() | 이름 순으로 정렬 |
| Void sortByID() | ID 순으로 정렬 |
| Void sortByScore() | 성적 순으로 정렬 |
| Student | Void print() | 학생 데이터 출력 |
| Student Manager | Void gen\_students() | 학생 10명 데이터 랜덤 생성 |
| Void Run() | 유저 인터페이스 출력 및 학생관리프로그램 실행 |
| FileIO | Static void save(const LinkedList<T>& slist) | 파일에 모든 학생데이터 저장 |
| Static void (oad(LinkedList<T>& slist) | 파일에서 학생데이터 불러오기 |

1. 최종 결과 및 추가 내용
   1. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
      1. 현재 프로젝트 설계(구현 내용)
         1. 현재 내장된 연결리스트는 특정 데이터 타입에 종속되지 않고 어떤 프로그램에서도 사용할 수 있도록 템플릿을 이용하여 작성되었음
         2. 선택 정렬 알고리즘을 사용하여 이름 순, ID 순, 성적 순으로 정렬이 가능하고, 동작 후 정렬된 내용을 화면에 출력함
         3. 이름과 ID에 해당하는 데이터를 검색하여 조회하고 삭제할 수 있음
         4. 사용자가 UI를 통해 원하는 기능을 입력했을 때 버퍼를 비우고 예외를 검사하여 프로그램이 예상치 못한 무한반복이나 잘못된 출력을 하지 않도록 함
      2. 향후 개발 내용
         1. 파일 입출력 시 데이터를 직렬화하고, 텍스트 파일 대신 바이너리 파일을 생성하여 데이터의 안정성 확보
         2. 현재 정렬 알고리즘은 선택 정렬 알고리즘을 통하여 구현하였으나 차후 데이터 양이 증가할 시를 대비해 퀵 정렬 알고리즘으로 변경
         3. 프로그램이 실행되었을 때 자동으로 무작위 학생 데이터를 생성하여 연결리스트에 저장되고 있기 때문에, 차후에는 저장된 파일을 불러오는 방식으로 변경하여 실사용 측면에서 더 나은 환경을 제공
         4. 코드에 불필요한 반복문을 줄이고 적절한 예외처리 등을 통해 프로그램의 안전성과 유지보수성을 증가
         5. StudentManager에 집중된 기능들과 밀집된 함수들을 기능별로 모듈화하여 열거형 변수를 관리하는 클래스, 정렬과 검색을 담당하는 클래스 등으로 세분화
         6. 학생 ID가 중복되지 않도록 하여 데이터의 신뢰성을 확보하고 동명이인의 데이터를 ID와 연계하여 관리함으로써 이름이 중복되더라도 검색 시 모든 데이터 출력
   2. 최종 결과 화면

