**게임개발자전문가과정**

**과 목 명 : 게임기초프로그래밍**

**능력단위 : C/C++ 프로그래밍**

**제출일자 : 2022년 07월 26일**

**포트폴리오 : 파일 입출력 및 템플릿을 적용한 학생관리 프로그램 제작**

**작 성 자 : 김동완**

**<제출내역>**

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

|  |
| --- |
|  |

**< Contents >**

1. **프로젝트 소개 및 개요**
2. 프로젝트 소개
3. 프로젝트 주요 기술
4. **프로젝트 설계 및 다이어그램**
5. 시퀀스 다이어그램
6. 클래스 다이어그램
7. 클래스 Docs
8. **최종 결과 및 추가 내용**
9. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
   1. 현재 프로젝트 설계(구현 내용)
   2. 향후 개발 내용
10. 최종 결과
11. **프로젝트 소개 및 개요**
12. **프로젝트 소개**

C++의 특징을 최대한 활용하여 연결 리스트(Template Linked List)에 대한 구조 및 주요 기능과 파일 입출력을 설계하여 학생 관리 프로그램을 설계 및 구현한다.

단일 연결 리스트가 아닌 이중 연결 리스트를 구현하며 재 사용성을 위해 Template Class로 구현한다. 해당 프로젝트에서는 STL의 List의 기능을 참조하여 기능을 구현하고 스택과 큐의 역할 또한 가능하게 한다.

고정적인 수가 아닌 N명의 데이터를 모두 관리 할 수 있어야 하며 데이터 관리를 편하게 할 수 있도록 파일 입출력을 이용하여 저장과 불러오기를 가능케 하며, Console UI를 구현하여 사용자가 더욱 편리하게 사용 할 수 있게 설계 및 구현한다.

1. **프로젝트 주요 기술**

* **Template Double Linked List**

C와 C++ STL의 Container들은 대부분 Array 기반으로 Index로 접근이 가능하여 편리하나 Container의 중간이나 처음 위치에 데이터를 삽입하기에는 어려움이 있다. 이러한 상황을 대비하여 Double Linked List를 구현하여 어느 위치에나 데이터 삽입을 용이하게 할 수 있으며, Template을 이용하여 STL의 List의 기능을 구현하여 재 사용성을 높여 다른 프로젝트에서도 사용 가능하게 한다.

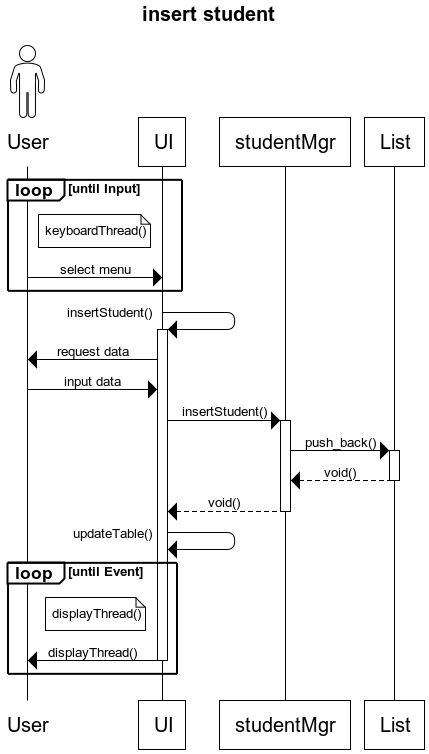
* **File I/O**

해당 프로젝트에서의 권장 사항은 블록 단위 입출력 사용이지만, 학생 관리 프로그램이라는 특성상 기밀성보단 편의성을 위하여 csv파일로 입출력 하도록 구현하였으며, 재 사용성은 고려하지 않았다.

* **Console UI**

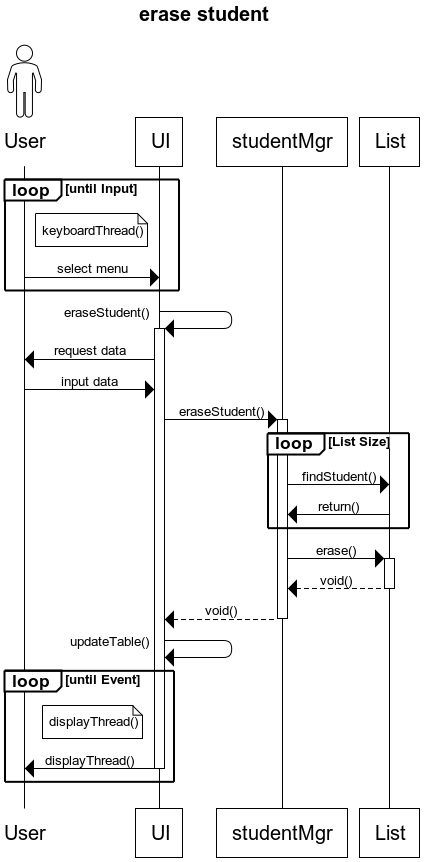
Win API를 이용하여 UI를 구성해 사용자의 편의성을 높였다.

1. **프로젝트 설계 및 다이어그램**
2. **시퀀스 다이어그램(Sequence Diagram)**
   1. **Insert Student**



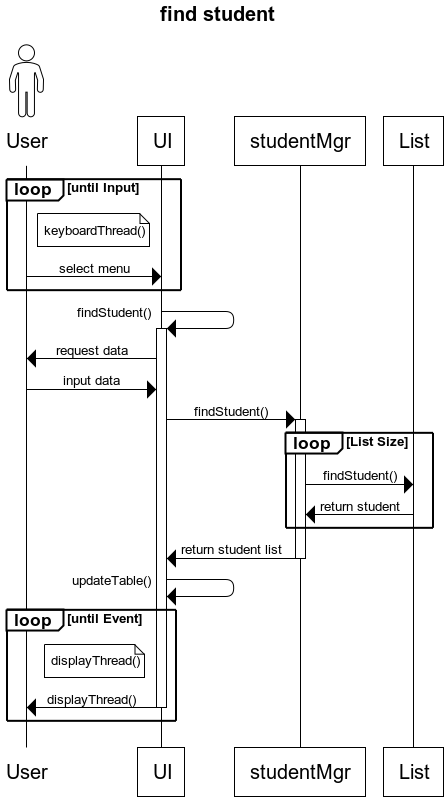
[그림 1-1] Insert 시퀀스 다이어그램(Sequence Diagram)

* 1. **Erase Student**

****

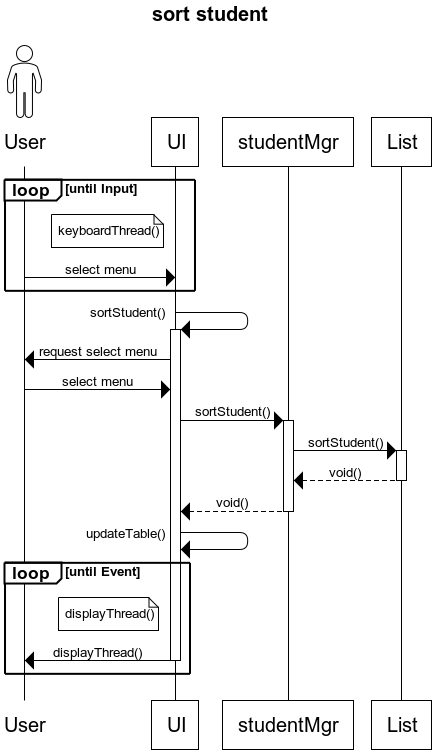
[그림 1-2] Erase 시퀀스 다이어그램(Sequence Diagram)

* 1. **Find Student**



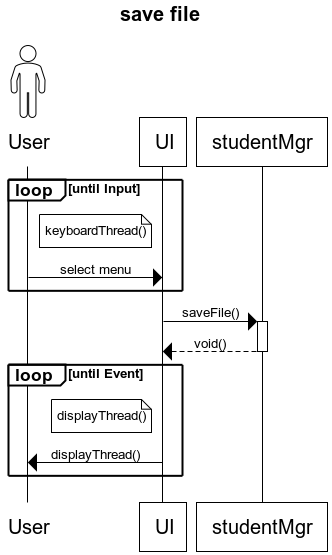
[그림 1-3] Find 시퀀스 다이어그램(Sequence Diagram)

* 1. **Sort Student**



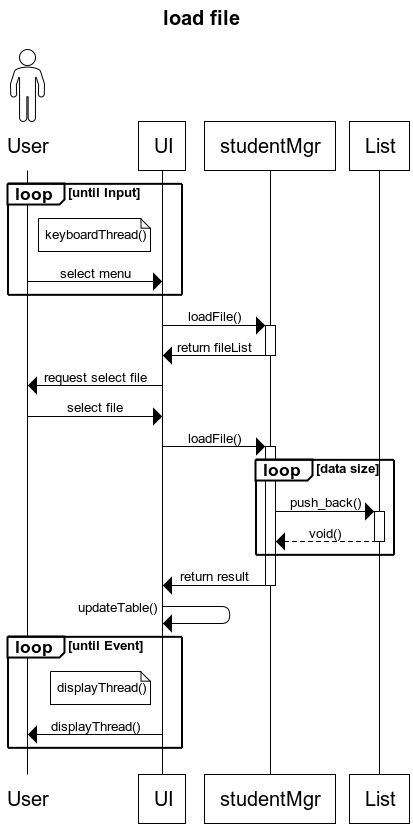
[그림 1-4] Sort 시퀀스 다이어그램(Sequence Diagram)

* 1. **Save File**



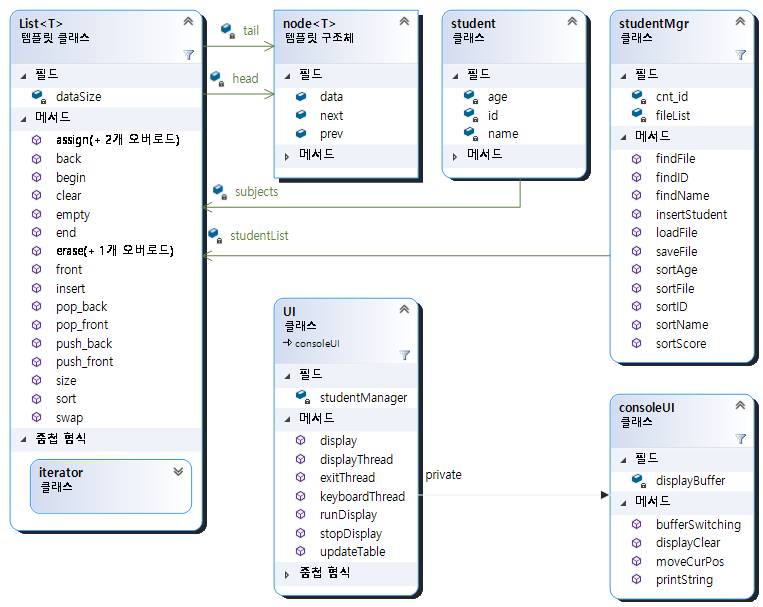
[그림 1-5] Save 시퀀스 다이어그램(Sequence Diagram)

* 1. **Load File**

****

[그림 1-6] load 시퀀스 다이어그램(Sequence Diagram)

1. **클래스 다이어그램(Class Diagram)**



[그림 1-7] 클래스 다이어그램(Class Diagram)

1. **클래스 Docs**

**Node Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| T data | 데이터 변수 |
| Node<T>\* prev | 이전 노드를 위한 포인터 변수 |
| Node<T>\* next | 다음 노드를 위한 포인터 변수 |

**List Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| Node<T>\* head | List의 처음을 가리키는 포인터 변수. |
| Node<T>\* tail | List의 끝을 가리키는 포인터 변수. |
| size\_t dataSize | List에 저장된 노드의 개수 |

**List Function**.

|  |  |
| --- | --- |
| 이름 | 설명 |
| void assign(int \_count, T& \_data = 0) | \_count: 생성 개수.  \_data: 데이터. (default = 0)  생성 개수만큼 데이터 참조 생성. |
| void assign(List& \_list) | \_list: 참조 리스트.  참조 리스트 복사 후 생성. |
| void assign(iterator \_first, iterator \_last) | \_first: 시작 위치 반복자.  \_end: 마지막 위치 반복자.  시작 위치부터 마지막 위치까지의 데이터 복사 후 생성. |
| T& back() | 마지막 요소 반환. |
| iterator begin() | 첫 요소 위치의 반복자 반환. |
| const iterator cbegin() | 첫 요소 위치의 const 반복자 반환. |
| iterator end() | 마지막 요소 다음 위치의 반복자 반환. |
| const iterator cend() | 마지막 요소 다음 위치의 const 반복자 반환. |
| void clear() | 리스트 내의 노드 전부 삭제. |
| void erase(iterator \_iter) | \_iter: 위치 반복자.  반복자 위치의 요소 삭제. |
| void erase(iterator \_first, iterator \_last) | \_first: 시작 위치의 반복자.  \_last: 마지막 위치의 반복자.  시작 위치부터 마지막 위치까지의 요소 삭제. |
| T& front() | 첫 요소 반환. |
| void insert(iterator \_iter, T& \_data) | \_iter: 위치 반복자.  \_data: 데이터.  반복자 위치에 데이터 삽입. |
| void pop\_back() | 마지막 요소 삭제. |
| void pop\_front() | 첫 요소 삭제. |
| void push\_back(T& \_data) | \_data: 데이터.  리스트 맨 뒤에 데이터 삽입. |
| void push\_front(T& \_data) | \_data: 데이터.  리스트 맨 앞에 데이터 삽입. |
| void swap(iterator \_iter1, iterator \_iter2) | \_iter1: 첫 번째 반복자.  \_iter2: 두 번째 반복자.  첫 번째 반복자 위치의 요소와 두 번째 반복자 위치의 요소 교환. |
| void sort(iterator \_first, iterator \_last, std::function<bool(T&, T&)> \_comparisonFunc) | \_first: 시작 위치 반복자.  \_last: 마지막 위치 반복자.  \_comparisonFunc: 비교 함수  시작 위치부터 마지막 위치까지의 데이터 정렬. |
| bool empty() | 리스트가 비어있는지 확인 후 반환. |
| size\_t size() | 리스트의 요소 개수 반환. |

**Student Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| int id | 고유 ID 변수. |
| std::string name | 이름 변수. |
| int age | 나이 변수. |
| List<subject> subjects | 과목과 점수를 저장하는 리스트. |

**studentMgr Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| int cnt\_id | 고유 ID 부여 횟수. |
| std::vector<std::string> fileList | 파일 이름 저장 컨테이너. |
| List<student> studentList | 학생 정보를 저장하는 리스트. |

**studentMgr Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| bool findFile(std::string \_extension = ".csv"); | \_extension: 확장자.  현재 위치에서 입력 받은 확장자의 파일 탐색. |
| bool saveFile(std::string \_name) | \_name: 파일 이름.  파일 생성 후 학생 정보 저장. (csv 포맷) |
| bool loadFile(std::string \_name) | \_name: 파일 이름.  파일에서 학생 정보 읽어 온 후 리스트에 삽입. |
| bool sortFile() | 파일 리스트 정렬. (Decending) |
| void sortID(bool \_ascending = true)  void sortName(bool \_ascending = true)  void sortAge(bool \_ascending = true) | \_ascending: 오름차순 플래그 (default: Ascending)  학생 정보의 ID, Name, Age에 따라 정렬. |
| void sortScore(ESUBJECT \_subject, bool \_ascending = false) | \_subject: 교과목.  \_ascending: 오름차순 플래그. (default: Decending)  교과목 성적에 따라 정렬. |
| List<student>::iterator findID(int \_id) | \_id: ID.  해당 ID의 반복자 위치 반환. |
| LL::List<student>::iterator findName(std::string \_name) | \_name: 이름.  해당 이름의 반복자 위치 반환. |
| bool findNameAll(std::string \_name, LL::List<student>& \_list) | \_name: 이름.  \_list: 찾기 결과 리스트  해당 이름의 학생을 모두 찾아 결과 리스트에 삽입. |

**consoleUI Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| HANDLE displayBuffer[2] | 더블 버퍼링을 위한 버퍼 핸들. |
| int bufferIdx | 버퍼 스위칭을 위한 인덱스. |

**consoleUI Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| void bufferSwitching() | 버퍼 스위칭. |
| void displayClear() | 버퍼 클리어. |
| void printString(int \_x, int \_y, std::string \_string) | \_x: column 위치.  \_y: row 위치.  \_string: 데이터.  해당 위치에 데이터 출력. (Line 출력) |
| void moveCurPos(int \_x, int \_y) | \_x: column 위치.  \_y: row 위치.  해당 위치로 커서 이동. |

**UI Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| POINT cursor | 커서 위치. |
| studentMgr\* studentManager | Student Manager 참조 위한 포인터 변수.. |
| HANDLE threadHandle[2] | display, keyboard 쓰레드를 위한 핸들. |
| HANDLE runEvent | display thread 구동을 위한 핸들. |
| HANDLE exitEvent | 프로그램 종료를 위한 핸들. |

1. **최종 결과 및 추가 내용**
2. **범용성/유연성/확장성/간결성 고려하여 추가된 내용**
   1. **현재 프로젝트 설계(구현 내용)**
3. 범용성, 유연성, 확장성, 재 사용성 모두를 고려하여 Template Double Linked List와 iterator를 함께 구현하여 STL의 List처럼 사용 가능.
4. Double buffering의 기능만을 가진 console UI 클래스를 상속받아 사용하여 재 사용성 높임.
5. 파일 입출력 시 결과를 쉽게 보기 위해 csv포맷을 채용하여 간결성을 높여 구현.
   1. **향후 개발 내용**
6. 현 프로젝트에서는 블록 단위 입출력을 사용하지 않았지만, 다른 프로젝트에서의 재사용을 위해 블록 단위 입출력 기능 추가 및 별개의 File I/O Class로 제작하여 재 사용성 높일 예정.
7. 시간 분배 미흡으로 UI Class의 경우 가시성이 좋지 못하여 재 사용이 어려울 것으로 예상되고, Menu Class의 상속을 통한 다형성 확보가 되지 않아 아쉬운 부분이 있어, 상속을 통해 각 Menu Class를 생성하여 다형성과 간결성 확보하며 전체적인 UI Class의 리펙토링 예정.
8. 몇 가지의 warning과 return 값을 반환하지 않는 경우가 있어, 예외 처리 및 형 변환 추가 예정.
9. 이미 입력된 데이터 수정 기능이 없어 리펙토링 시 구현 예정.
10. **최종 결과**

**텍스트이(가) 표시된 사진

자동 생성된 설명**