**게임개발자전문가과정**

**과 목 명 : 게임기초프로그래밍**

**능력단위 : C/C++ 프로그래밍**

**제출일자 : 2022년 07월 26일**

**포트폴리오 : 파일 입출력 및 템플릿을 적용한 종합 DB 프로그램 제작**

**작 성 자 : 신승빈**

**<제출내역>**

1. **종합 DB 프로그램 구현 프로젝트**
2. **종합 DB 프로그램 구현 분석 및 세부 문서**

|  |
| --- |
|  |

**< Contents >**

1. **프로젝트 소개 및 개요**
2. 프로젝트 소개
3. 프로젝트 주요 기술
4. **프로그램 실행 절차**
5. **프로젝트 설계 및 다이어그램**
6. 시퀀스 다이어그램
7. 클래스 다이어그램

2-1) 프로젝트에서 사용된 변수 역할

2-2) 프로젝트에서 사용된 함수 역할

1. **최종 결과 및 추가 내용**
2. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
   1. 현재 프로젝트 설계(구현 내용)
   2. 향후 개발 내용
3. **프로젝트 소개 및 개요**
4. **프로젝트 소개**

이번 프로젝트는 파일 입출력 및 템플릿을 적용하여 종합 DB 프로그램을 제작했습니다. 프로젝트를 통해 C/C++의 기능을 복기하고, 데이터를 텍스트 파일에 저장하고 읽어오는 것을 숙달했습니다. DB의 정보를 관리할 때 사용되는 자료구조로는 연결리스트를 사용하여 자료의 추가 및 삭제를 유연하게 수행할 수 있도록 구현했습니다. 또한 객체지향적 프로그래밍을 위해 독립적인 객체를 만들기 위해 설계에 많은 노력을 기울였습니다.

1. **프로젝트 주요 기술**

* **버퍼**

C++는 string이라는 클래스를 통해 데이터 배열을 쉽게 다루지만, C의 경우 데이터 배열을 위한 특별한 자료형이 없이 char 배열을 통해 데이터 배열을 다룹니다. 하지만 입출력될 데이터 배열의 크기는 미리 알 수 없고, 동적 할당은 스트림 입출력을 위한 크기 정보가 없기 때문에, char 배열만으로는 데이터 입출력을 다루기 어렵습니다. 이 문제를 해결하기 위해, Buffer라는 Custom클래스를 구현하여 자료형에 대한 관리를 담당하도록 하였습니다.

* **입출력**

모든 데이터의 전송은 Stream을 통해 표현이 됩니다. 다시 말해, 데이터 송수신의 주체가 User가 되었든, File이 되었든 송수신은 Stream을 통해 추상화가 되기 때문에 프로그램은 그 너머에 무엇이 있는지는 신경쓰지 않아도 됩니다. 이러한 Stream의 특성을 이용하기 위해, IOManager를 만들어 모든 데이터 입출력을 담당하게 만들었습니다. 다만, stdin 스트림의 경우 일반적인 입력으로 EOF를 받기 힘들기 때문에, 특별히 ‘\n’으로도 입력을 마무리 할 수 있도록 구현했습니다.

* **파일**

모든 파일은 FILE 포인터를 통해 관리되며, 파일의 open, close는 반드시 수행되어야 하고, 파일을 읽기, 쓰기, 수정 작업은 open의 mode 설정에 따라 섬세하게 수행됩니다. 그렇기 때문에, 이 복잡한 모든 절차들의 책임을 FileManager 클래스에 위임하여 사용자는 이를 특별히 신경쓰지 않도록 설계하였습니다.

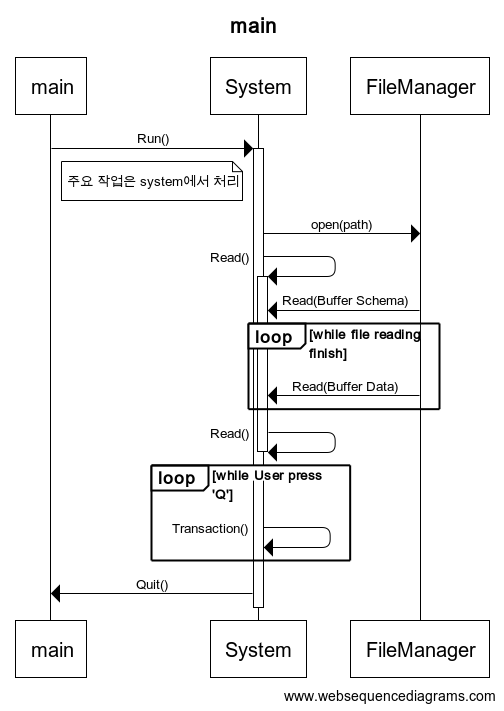
* **데이터와 스키마**

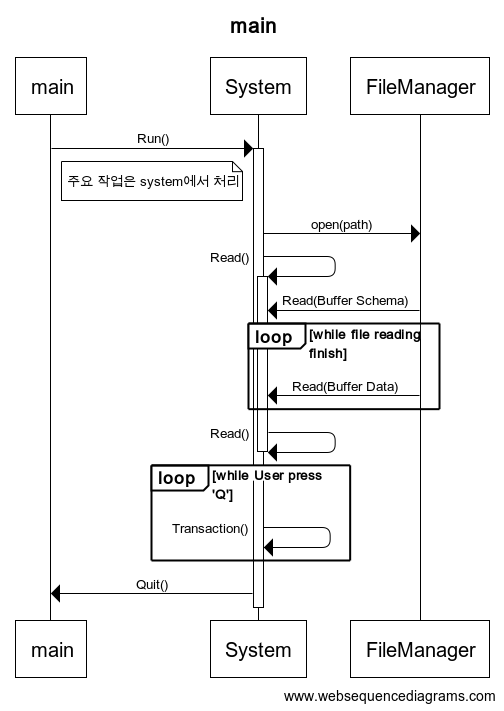
특정한 형식이 없는 종합 DB 시스템의 경우 저장될 데이터 필드가 특별히 정해지지 않았고, 그 필드의 생성과 삭제 또한 자유롭기 때문에 데이터 필드 정보를 저장하기 위한 스키마 클래스를 따로 구현했습니다. 데이터 클래스는 이런 스키마 클래스를 통해 자신이 가지고 있는 불특정한 데이터 배열을 유의미하게 해석할 수 있습니다.

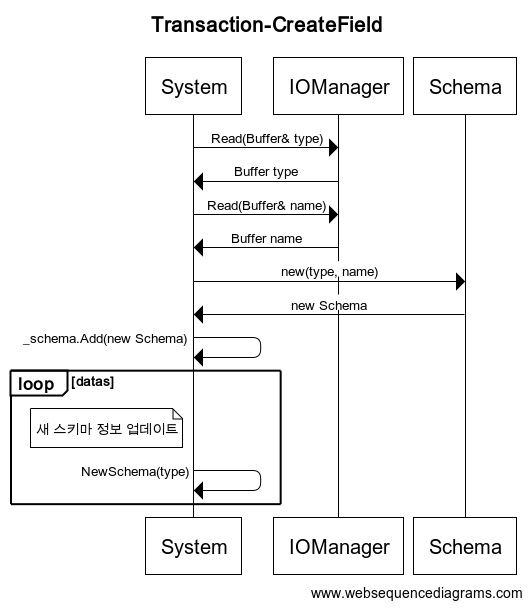
* **템플릿 연결리스트**

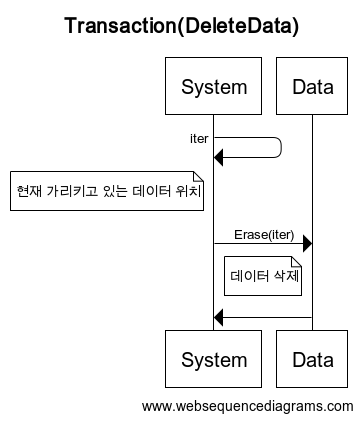
객체 지향성을 살리기 위해 연결 리스트를 템플릿을 통해 구현하고, 템플릿 내부에는 Node 클래스를 통해 리스트의 노드간 이동을 담당하도록 구현했습니다. 또한, 리스트의 이동을 추상화하기 위해 Iterator패턴도 구현하여 노드간의 이동을 간편화 했습니다. 연결리스트는 라이브러리화 하여 이전에 배운 라이브러리 생성 절차도 수행했습니다.

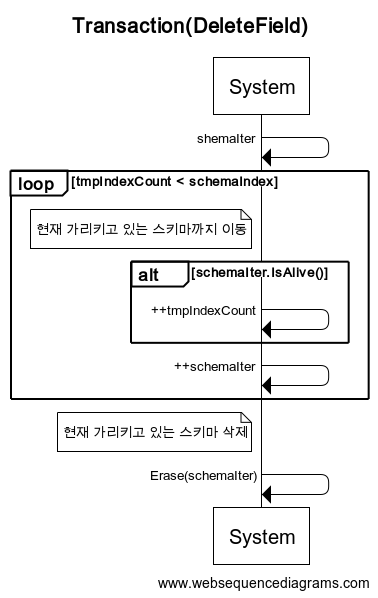
1. **프로그램 실행 절차**
   1. 읽을 파일명에 확장자를 제외한 파일명을 작성
      1. 파일이 없을 경우 파일 생성.
      2. 파일이 있을 경우 파일 읽음
   2. 스키마가 비어있을 경우 스키마부터 생성해야 데이터 저장 가능
   3. 스키마가 생성되었다면 Default로 초기화된 데이터 생성 가능
   4. vim기반으로 이동을 지원하며 Carret이 위치한 곳에서 e를 눌러 데이터 수정 가능
   5. Carret이 위치한 곳에서 스키마 또는 데이터 삭제 가능
   6. s를 눌러 데이터 저장 가능
   7. q를 눌러 프로그램 종료
2. **프로젝트 설계 및 다이어그램**
3. **시퀀스 다이어그램(Sequence Diagram)**

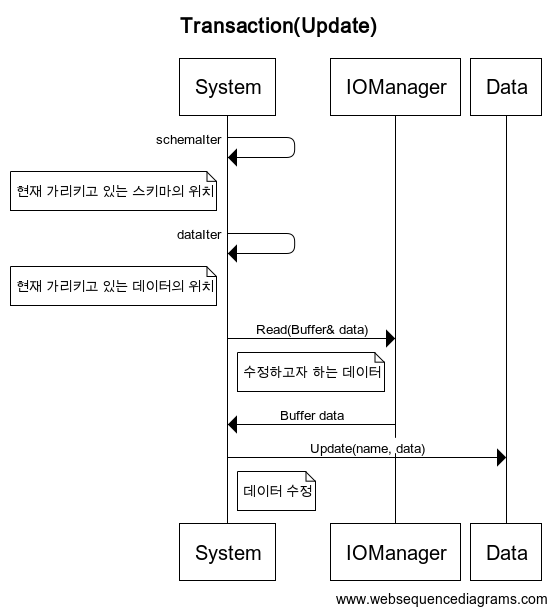


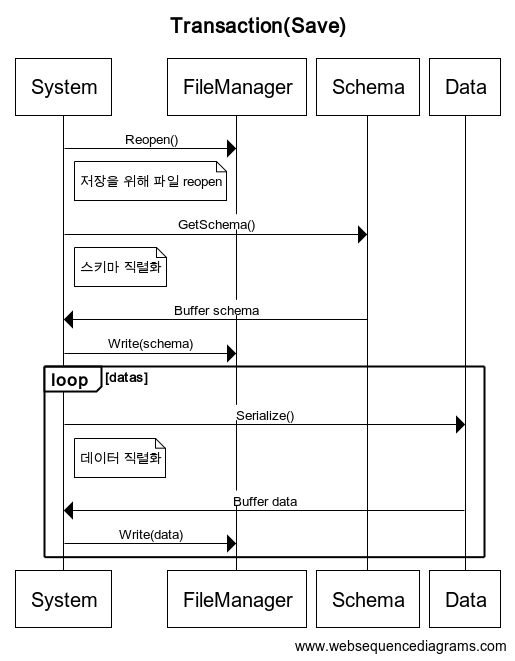






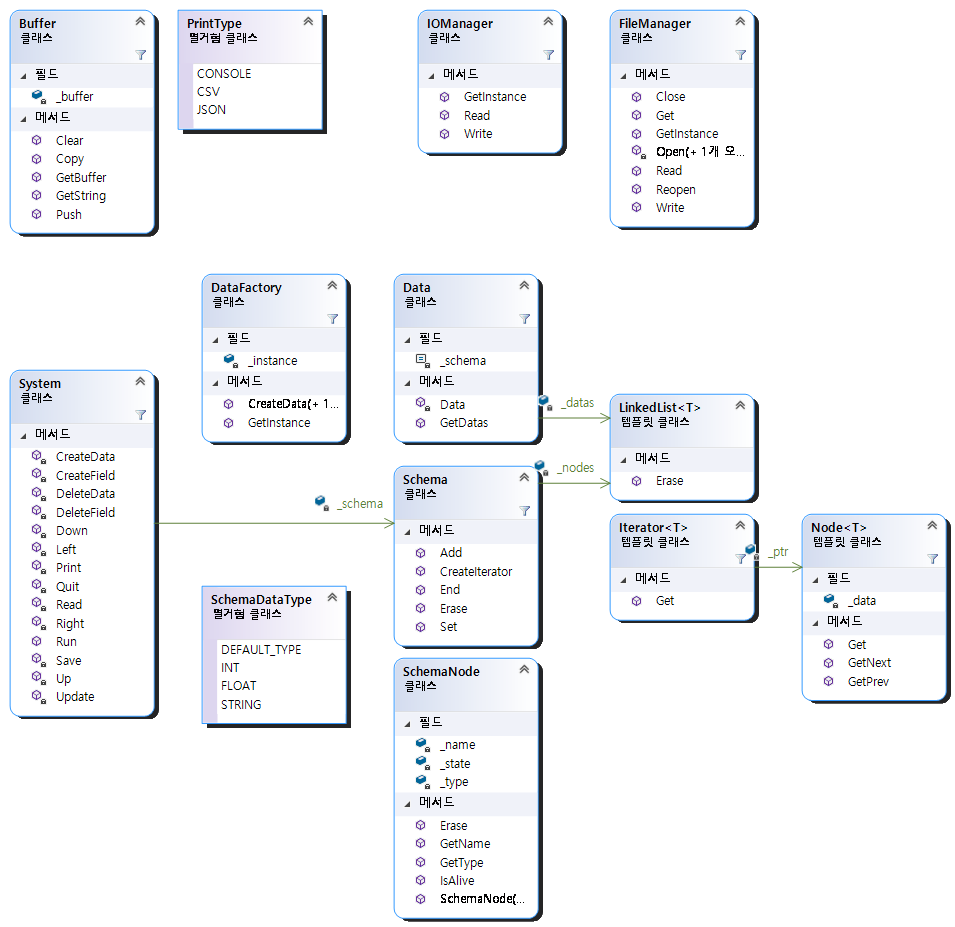






[그림 1-1] 시퀀스 다이어그램(Sequence Diagram)

1. **클래스 다이어그램(Class Diagram)**



[그림 1-2] 클래스 다이어그램(Class Diagram)

**2-1) 프로젝트에서 사용된 변수 역할**

|  |  |  |
| --- | --- | --- |
| **클래스명** | **변수** | **내용** |
| **Node** | T\* \_data | 노드에 저장할 데이터 변수 |
| Node\* \_pNext | 다음 노드를 가리키기 위한 노드 포인터 변수 |
| Node\* \_pPrev | 이전 노드를 가리키기 위한 노드 포인터 변수 |
| **Iterator** | Node<T>\* \_ptr | Iterator가 가리키는 노드를 지칭하는 포인터 변수 |
| **Linked**  **List** | Node<T>\* \_pHead | 연결리스트의 첫번째 노드를 가리키는 노드 포인터 변수 |
| Node<T>\* \_pTail | 연결리스트의 마지막 노드를 가리키는 노드 포인터 변수 |
| **Buffer** | CUSTOM\_BYTE\* \_buffer | 데이터 배열 |
| Int \_maxSize | 버퍼의 최대 크기 |
| Int \_currSize | 버퍼 내부에 저장된 데이터 크기 |
| **Data** | LinkedList<Buffer> \_datas | 데이터 객체가 스키마에 의해 갖는 필드별 데이터 값 |
| Const Schema\* const \_schema | 원본 스키마 값 |
| **SchemaNode** | SchemaDataType \_type | 입력받는 값의 타입을 나타내는 값 |
| Char \_name[NameLength] | 필드의 이름 |
| Int \_state | 스키마가 삭제되었는지 알려주는 값 |
| **Schema** | LinkedList<SchemaNode> \_nodes | 스키마 필드를 저장하는 값 |
| **FileManager** | FILE\* \_fp | 파일을 가리키는 값 |
| Buffer \_path | 파일 경로를 저장한 값 |
| **IOManager** | static IOManager\* \_instance | singleton으로 구현하기 위한 값 |

[표 1-1] 프로젝트의 클래스 변수

**2-2) 프로젝트에서 사용된 함수 역할**

|  |  |  |
| --- | --- | --- |
| **클래스명** | **함수** | **내용** |
| **Node** | void SetData(const T& data) | 노드의 데이터를 복사 저장하는 함수 |
| void SetData(T&& data) | 노드의 데이터를 이동 저장하는 함수 |
| T& Get() | 노드가 담고있는 데이터를 반환하는 함수 |
| **Iterator** | T& Get() | Iterator가 포인팅하는 Node가 담고있는 데이터를 반환하는 함수 |
| **LinkedList** | Void Link(Node<T>\* prev, Node<T>\* curr, Node<T>\* next) | curr을 중심으로 prev와 next를 연결하는 함수. |
| Void unLink(Node<T>\* prev, Node<T>\* curr, Node<T>\* next) | curr을 제외하고 prev와 next를 연결하는 함수. |
| Void PushBack(const T& data) | 리스트 뒤에 데이터를 복사 저장하는 함수 |
| Void PushBack(T&& data) | 리스트 뒤에 데이터를 이동 저장하는 함수 |
| Void PushFront(const T& data) | 리스트 앞에 데이터를 복사 저장하는 함수 |
| Void PushFront(T&& data) | 리스트 앞에 데이터를 이동 저장하는 함수 |
| Void Erase(Iterator<T>& iter) | Iterator가 지정한 노드를 삭제하는 함수 |
| Void EraseBack() | 마지막 노드를 삭제하는 함수 |
| Void EraseFront() | 처음 노드를 삭제하는 함수 |
| **Buffer** | Void Copy(const CUSTOM\_BYTE\* pArr, int size) | 지정된 크기만큼 버퍼에 데이터를 복사하는 함수 |
| Void Push(const CUSTOM\_BYTE\* pArr, int size) | 지정된 크기만큼 버퍼에 데이터를 추가하는 함수 |
| Const char\* GetString | C Style string으로 반환하는 함수 |
| Const CUSTOM\_BYTE\* GetBuffer() | 버퍼에 저장된 데이터 배열을 반환하는 함수 |
| **Data** | Void Parallelize(const Buffer& serialData) | 직렬화된 데이터를 분리하여 Data객체에 Schema 필드별로 저장하는 함수 |
| Void Update(Buffer fieldName, Buffer data) | 데이터를 수정하는 함수 |
| Buffer Serialize(bool print) | print가 참일 경우 console출력을 위해 직렬화 하는 함수  print가 거짓일 경우 저장을 위해 직렬화하는 함수 |
| **DataFactory** | Data\* CreateData(const Schema\* schema, const Buffer& serializedBuffer) | Data 객체를 생성해 반환하는 함수  주어진 스키마에의해 직렬화된 데이터인 serializedBuffer값을 분리하여 저장 |
| **SchemaNode** | void Erase() | 스키마 필드를 삭제하는 함수 |
| bool IsAlive() | 스키마 필드가 삭제되었는지 확인하는 함수 |
| **Schema** | void Add(SchemaDataType type, const char\* name) | 스키마 필드를 추가하는 함수 |
| void Erase(Iterator<SchemaNode>& iterator) | 스키마 필드를 삭제하는 함수 |
| **FileManager** | bool Read(Buffer& buffer) | 파일로부터 읽어서 buffer에 저장하여 반환하는 함수 |
| void Write(Buffer& buffer) | buffer에 저장된 값을 파일에 작성하는 함수 |
| **IOManager** | static IOManager& GetInstance() | singleton으로 구현된 클래스를 호출하는 함수 |
| void Read(Buffer& buffer, FILE\* fp = stdin, int count = 0) | 스트림으로부터 읽어온 값을 buffer에 저장하여 반환하는 함수 |
| void Write(Buffer& buffer, FILE\* fp) | buffer에 저장된 값을 스트림으로 출력하는 함수 |

[표 1-2] 프로젝트의 클래스 함수

1. **최종 결과 및 추가 내용**
2. **범용성/유연성/확장성/간결성 고려하여 추가된 내용**
   1. **현재 프로젝트 설계(구현 내용)**
3. 현재 프로젝트는 범용적인 DB시스템을 위한 프로그램을 위해 스키마와 데이터가 분리되어 설계되었음.
4. 또한 파일에 읽고 쓰는 것을 관리하는 FileManager와 스트림에서의 데이터 입출력을 담당하는 IOManager를 분리하여 File이 아닌 stdin이 입출력을 수행하더라도 정상적으로 작동함.
5. 데이터를 관리하는 Buffer를 따로 분리하여 사용자의 데이터 상태 및 크기를 신경쓰지 않도록 구현함.
6. DB시스템의 가장 기본적인 기능인 CRUD가 구현되어있음.
   1. **향후 개발 내용**
7. 초기 계획했던 Memory Pool을 이용한 데이터 관리
8. Thread를 이용한 파일 Loading
9. 데이터 타입에 따른 적합한 데이터 입력 점검
10. 지정된 스키마를 기준으로 하는 정렬
11. 사용자가 입력한 데이터와 일치하는 Data들 검색 및 출력
12. 테이블간의 연산 함수 지원