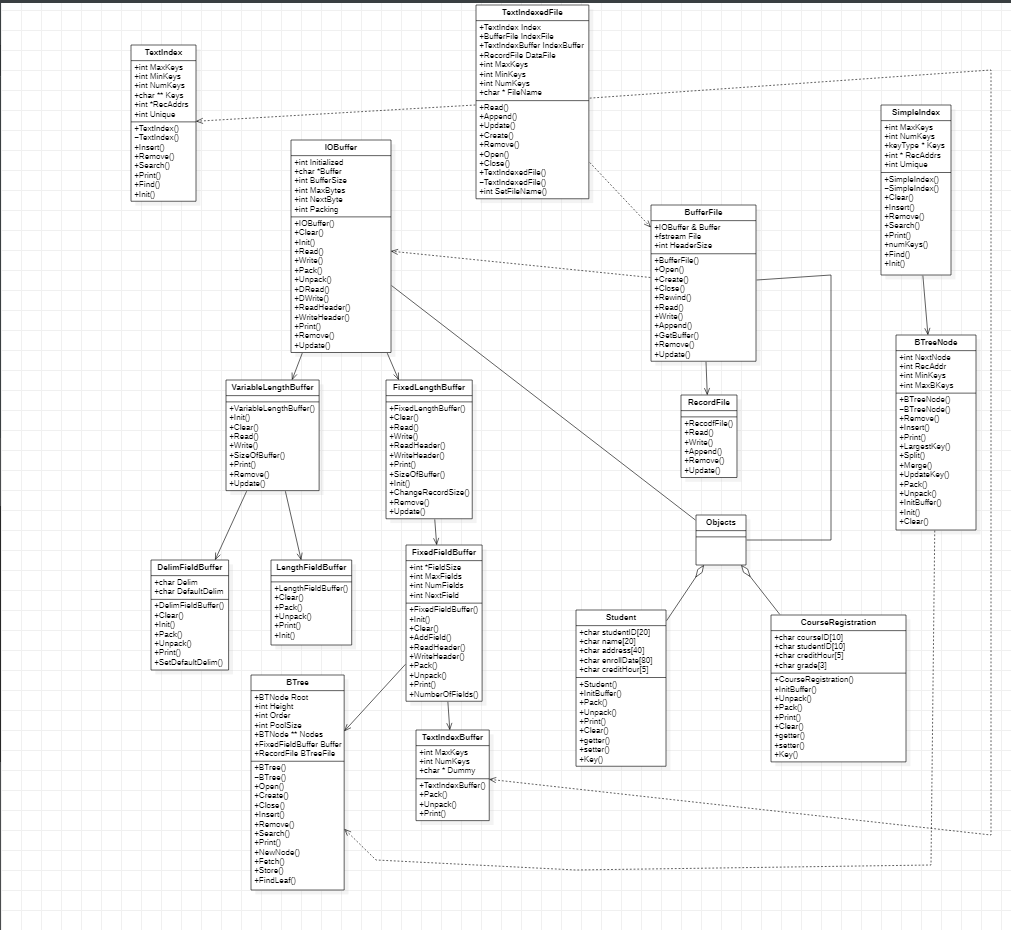
|  |
| --- |
| HW5. **B-Tree**  Department Name: 정보컴퓨터공학부  Student Number: 201424528  Github ID: jsg6420  Name: 정승규  Submission date: 2020-06-21  Course: File Structure (060) |

The report should include the following:

# Class design

1. UML class diagram



1. Explanations for each class

- What are the member variables and member functions.

1. IOBuffer

* int initialized: buffer가 초기화 되면 True이다.
* char \*Buffer: 필드 값을 가지고 있는 캐릭터 배열이다.
* Int BufferSize: Pack된 필드들의 총 사이즈
* int MaxBytes: buffer에서 가장 큰 bytes
* int NextBytes: 다음에 Pack/Unpack될 byte의 index
* int Packing: Pack일 때 1, Unpack일 때 0
* IOBuffer(): IOBuffer 클래스의 생성자이다.
* Clear(): buffer의 필드 값을 clear한다.
* Init(): buffer를 초기화한다.
* Read(): buffer로부터 데이터를 읽는다.
* Write(): buffer에 데이터를 저장한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* DRead(): Read operation에 direct로 접근한다.
* DWrite(): Write operation에 direct로 접근한다.
* ReadHeader(): 파일의 header를 읽는다.
* WriteHeader(): 파일에 header를 작성한다.
* Remove(): 데이터를 제거한다.
* Update(): 데이터를 최신화한다.

1. VariableLengthBuffer

* VariableLengthBuffer(): VariableLengthBuffer 클래스의 생성자.
* Init(): buffer를 초기화한다.
* Clear(): buffer의 필드 값을 clear한다.
* Read(): buffer로부터 데이터를 읽는다.
* Write(): buffer에 데이터를 저장한다.
* SizeOfBuffer(): buffer의 현재 크기를 리턴한다.
* Print(): buffer 데이터를 출력한다.
* Remove(): 데이터를 제거한다.
* Update(): 데이터를 최신화한다.

1. FixedLengthBuffer

* FixedLengthBuffer(): FixedLengthBuffer 클래스의 생성자
* Clear(): buffer의 필드 값을 clear한다.
* Read(): buffer로부터 데이터를 읽는다.
* Write(): buffer에 데이터를 저장한다.
* ReadHeader(): 파일의 header를 읽는다.
* WriteHeader(): 파일에 header를 작성한다.
* Print(): buffer 데이터를 출력한다.
* SizeOfBuffer(): buffer의 현재 크기를 리턴한다.
* Remove(): 데이터를 제거한다.
* Update(): 데이터를 최신화한다.

1. DelimFieldBuffer

* Char Delim: 필드 데이터를 구분할 구분자
* Char DefaultDelim: 기본 구분자
* DelimFieldBuffer(): DelimFieldBuffer 클래스의 생성자
* Clear(): buffer의 필드 값을 clear한다.
* Init() : buffer를 초기화한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): buffer 데이터를 출력한다.
* SetDefaultDelim(): 기본 구분자를 설정한다.

1. LengthFieldBuffer

* LengthFieldBuffer(): LengthFieldBuffer 클래스의 생성자
* Clear(): buffer의 필드 값을 clear한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): buffer 데이터를 출력한다.
* Init(): buffer를 초기화한다.

1. FixedFieldBuffer

* int \* FieldSize: 필드 사이즈를 저장하는 배열
* int MaxFields: 최대 필드 수
* int NumFields: 정의된 필드의 실제 수
* int NextField: 다음에 Pack/Unpack될 필드의 index
* FixedFieldBuffer(): FixedFieldBuffer 클래스의 생성자
* Clear(): buffer의 필드 값을 clear한다.
* Init(): buffer를 초기화한다.
* AddField(): 저장될 필드를 추가한다.
* ReadHeader(): 파일의 header를 읽는다.
* WriteHeader(): 파일의 header를 작성한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): buffer 데이터를 출력한다.
* NumberOfFields(): 정의된 필드의 수를 리턴한다.

1. BufferFile

* IOBuffer & Buffer: 데이터 저장 시 사용될 buffer의 종류
* Fstream File: 데이터를 저장 할 파일
* int HeaderSize: 헤더의 크기
* BufferFile(): BufferFile 클래스의 생성자
* Open(): 존재하는 파일을 open한다.
* Create(): 새로운 파일을 생성한다.
* Close(): 열려 있는 파일을 닫는다.
* Rewind(): 첫 번째 데이터 레코드로 재설정한다.
* Read(): buffer에 레코드를 읽는다.
* Write(): 현재 buffer의 데이터를 write한다.
* Append(): 현재 buffer를 파일의 끝에 write한다.
* GetBuffer(): 사용중인 Buffer를 리턴한다.
* Remove(): 데이터를 제거한다.
* Update(): 데이터를 최신화한다.

1. RecoedFile

* Read(): buffer에 레코드를 읽는다.
* Write(): 현재 buffer의 데이터를 write한다.
* Append(): 현재 buffer를 파일의 끝에 write한다.
* RecordFile(): RecordFile 클래스의 생성자
* Remove(): 데이터를 제거한다.
* Update(): 데이터를 최신화한다.

1. Student

* Char studentID[10]: 학번
* Char name[20]: 이름
* Char address[40]: 주소
* Char enrollDate[80]: 등록 일자
* Char creditHour[5]: 학점
* Student(): Student 클래스의 생성자
* InitBuffer(): 어떤 버퍼를 사용할 지 결정한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): buffer 데이터를 출력한다.
* Clear(): buffer의 필드 값을 clear한다.
* Getter(): 각 멤버 변수의 값을 리턴한다.
* Setter(): 각 멤버 변수의 값을 설정한다.
* Key(): 클래스의 key를 생성한다.

1. CourseRegistration

* Char courseID[10]: 과목 ID
* Char studentID[10]: 학번
* Char creditHour[5]: 학점
* Char grade[3]: 점수
* CourseRegistration(): CourseRegistration 클래스의 생성자
* InitBuffer(): 어떤 버퍼를 사용할 지 결정한다.
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): buffer 데이터를 출력한다.
* Clear(): buffer의 필드 값을 clear한다.
* Getter(): 각 멤버 변수의 값을 리턴한다.
* Setter(): 각 멤버 변수의 값을 설정한다.
* Key(): 클래스의 Key를 생성한다.

1. TextIndexedFile

* TextIndex Index: TextIndexedFile에 사용될 index
* BufferFile IndexFile: index를 저장할 파일
* TextIndexBuffer IndexBuffer: index buffer
* RecordFile DataFile: 데이터가 저장되어 있는 파일
* int MaxKeys: 저장될 수 있는 최대 index 수
* int MinKeys: 저장될 수 있는 최소 index 수
* int NumKeys: 현재 저장되어 있는 index 수
* Read(): 데이터를 읽는다.
* Append(): 데이터와 index를 추가한다.
* Update(): 파일에 저장되어 있는 데이터를 수정한다.
* Create(): index 파일을 생성한다.
* Remove(): 데이터와 index를 제거한다.
* Open(): index 파일을 open 한다.
* Close(): index 파일을 닫는다.
* SetFileName(): 데이터파일, index 파일 이름을 설정한다.
* TextIndexedFile(): TextIndexedFile 클래스의 생성자
* ~TextIndexedFile(): TextIndexedFile 클래스의 소멸자

1. TextIndex

* int MaxKeys: 저장될 수 있는 최대 index 수
* int MinKeys: 저장될 수 있는 최소 index 수
* int NumKeys: 현재 저장되어 있는 index 수
* char \*\* Keys: key 값의 배열
* int \*RecAddrs: record가 저장되어 있는 reference의 집합
* int Unique: 1이면 각 key 값은 index에서 unique해야 한다.
* TextIndex(): TextIndex 클래스의 생성자
* ~TextIndex(): TextIndex 클래스의 소멸자
* Insert(): index를 index 파일에 삽입한다.
* Remove(): index를 index 파일에서 제거한다.
* Search(): key 값을 바탕으로 record를 검색한다.
* Print(): index들을 출력한다.
* Find(): key 값을 바탕으로 record가 저장되어 있는 위치를 찾는다.
* Init(): TextIndex를 초기화한다.

1. TextIndexBuffer

* int MaxKeys: 저장될 수 있는 최대 index 수
* int KeySize: key의 크기
* char \* Dummy: pack, unpack 시 비어 있는 공간에 채울 Dummy
* TextIndexBuffer(): TextIndexBuffer 클래스의 생성자
* Pack(): buffer의 다음 필드 값을 설정한다.
* Unpack(): buffer의 다음 필드 값을 추출한다.
* Print(): 버퍼의 내용을 출력한다.

1. SimpleIndex

* int MaxKeys; 최대 key 개수
* int NumKeys; 현재 key의 개수
* keyType \* Keys; key 집합
* int \* RecAddrs; key의 record address
* int Unique 1이면, 각 key가 index에서 unique 해야한다.
* SimpleIndex() SimpleIndex의 생성자
* ~SimpleIndex() SimpleIndex의 소멸자
* Clear() index의 모든 key를 삭제한다.
* Insert() 새 key를 저장한다.
* Remove() 특정 key를 삭제한다.
* Search() key값을 통해 record를 검색한다.
* Print() index를 출력한다.
* numKeys() 현재 key의 개수를 리턴한다.
* Find() keyType \* Keys에서 원하는 key의 위치를 찾는다.
* Init() index를 초기화한다.

1. BTreeNode

* int NextNode 같은 레벨에 있는 다음 노드의 주소
* int RecAddr 해당 노드의 BTree file 에서의 주소
* int MinKeys 한 노드에서 최소로 가져야 하는 key의 개수
* int MaxBKeys 한 노드에서 최대로 가질 수 있는 key의 개수
* BTreeNode() BTreeNode의 생성자
* ~BTreeNode() BTreeNode의 소멸자
* Insert() 노드에 key를 삽입한다.
* Remove() 노드에서 key를 삭제한다.
* Print() SimpleIndex의 Print를 호출한다.
* LargestKey() 해당 노드에서 가장 큰 key를 리턴한다.
* Split() Insert할 때 노드가 꽉 차면 Split 해야한다.
* Merge() Remove할 때 한 노드의 key가 MinKeys보다 작으면 Merge해야한다.
* UpdateKey() Insert나 Remove 시 부모의 key를 Update 해야 할 때 사용한다.
* Pack() buffer의 다음 필드 값을 설정한다.
* Unpack() buffer의 다음 필드 값을 추출한다.
* Init() 노드를 초기화한다.
* Clear() 노드에 있는 key를 모두 제거한다.

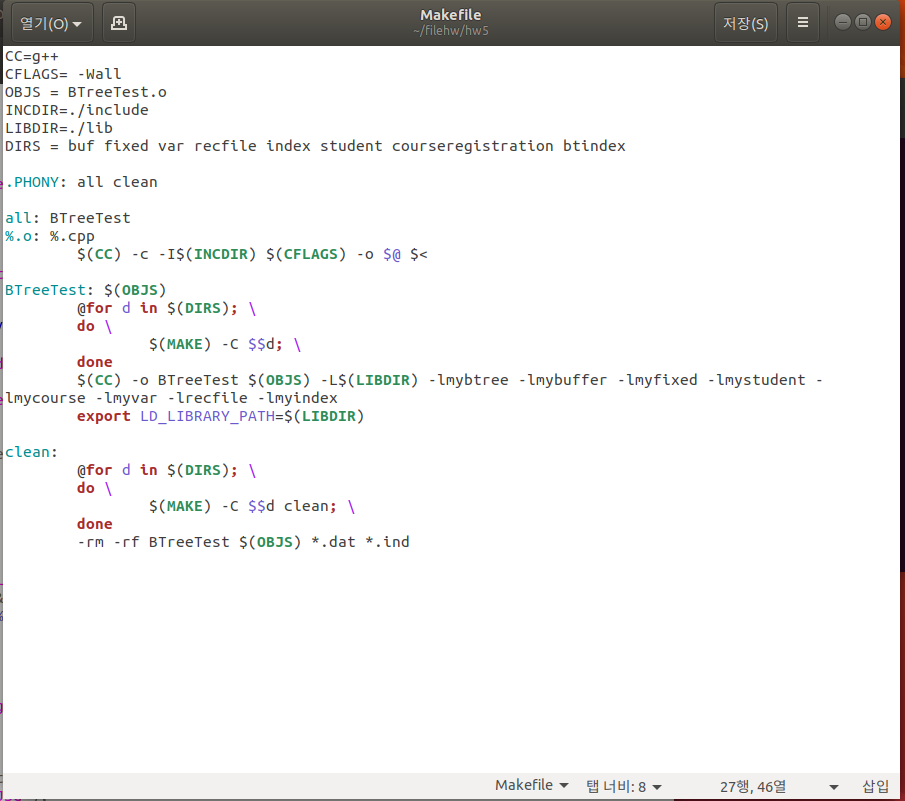
1. BTree

* BTNode Root;
* int Height Tree의 높이
* int Order Tree의 각 노드의 최대 filed 수
* int PoolSize; 최대 노드의 개수
* BTNode \*\* Nodes 사용 가능한 노드 집합
* FixedFieldBuffer Buffer 고정 길이 필드 버퍼
* RecordFile<BTNode> BTreeFile record를 저장할 파일
* BTree() BTree 클래스의 생성자
* ~BTree() BTree 클래스의 소멸자
* Open() data 파일을 open한다.
* Create() data 파일을 생성한다.
* Close() 파일을 닫는다.
* Insert() key를 BTree에 삽입한다.
* Remove() BTree에서 파라미터로 받은 key값을 삭제한다.
* Search() key를 검색한다.
* Print() Tree를 출력한다.
* FindLeaf() key가 속해 있는 노드를 찾는다.
* NewNode() Split할 때 새 노드를 생성하여 사용한다.
* Fetch() 노드의 주소로 노드의 정보를 가져온다.
* Store() 노드를 파일에 저장한다.

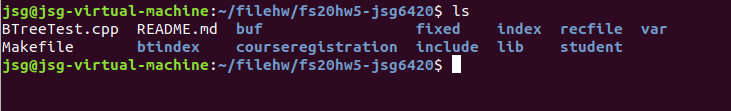
# Environment

1. Include Makefile

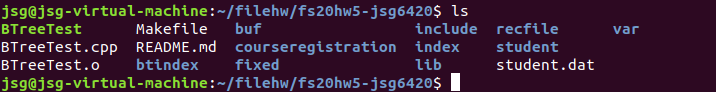
* Describe how to compile and execute you program



* Include 폴더에는 헤더 파일들이 있고, 나머지 폴더에는 각 소스 파일이 들어 있다. 그리고 각 폴더에는 Makefile이 개별적으로 존재한다. TestBTree라는 실행 파일을 만들기 위해서 먼저 TestBTree.cpp를 컴파일 하고 각 폴더에서 make를 수행하고 shared library로 so파일을 생성해서 lib에 저장한다. 그 후 마지막으로 so파일들과 TestBTree.o 파일을 linking 하여 실행 파일을 생성한다.

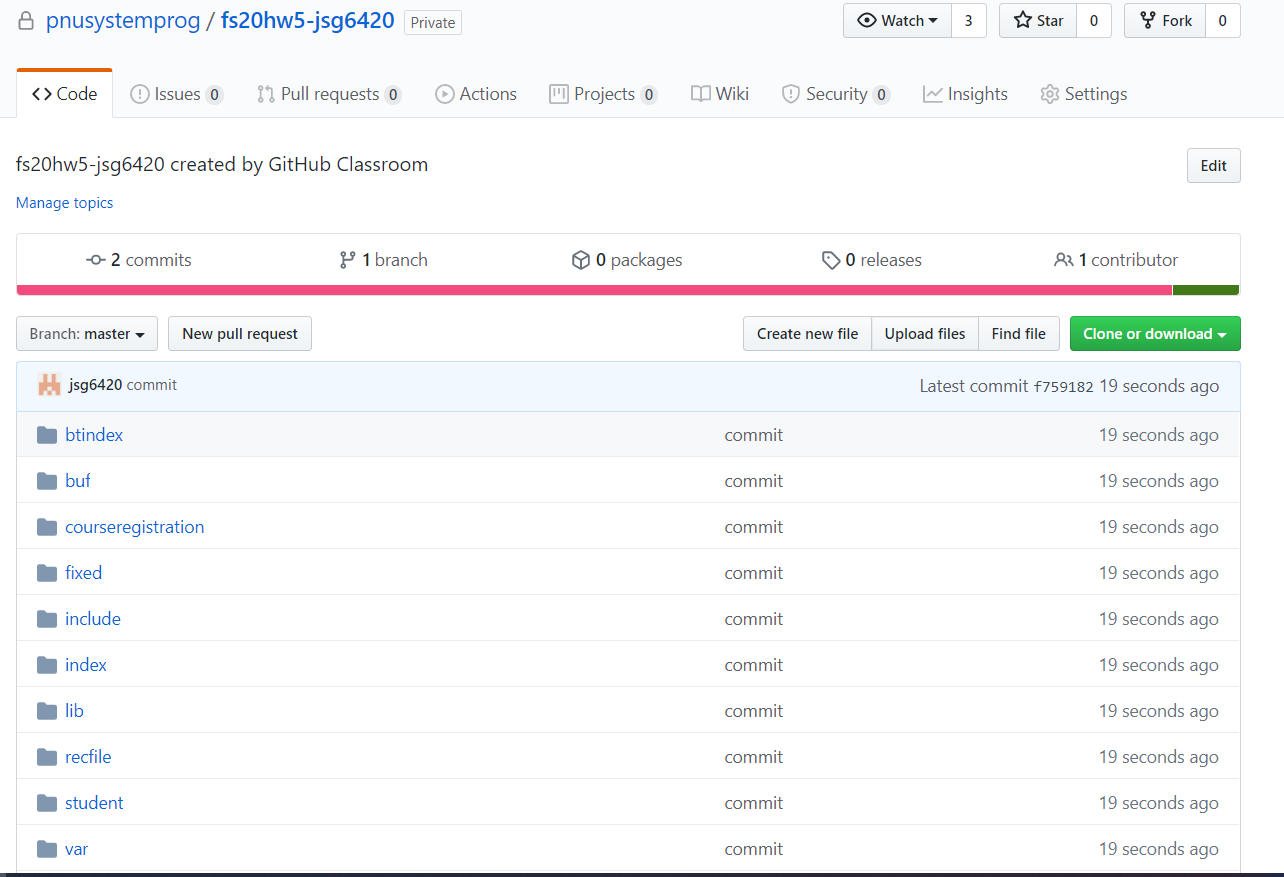


* Before compile.



* After compile and execution.
* Executable file IndexTest and DataFiles and IndexFiles are created.

1. Include a screen shot of your GitHub repository



# Test Program

* Explain how your system works perfectly according to the specification.

Btree에서 key를 삭제할 때는 몇 가지 경우를 따져봐야 합니다.

1. 삭제하려는 key가 해당 노드에서 가장 큰 값일 때

스크린샷이(가) 표시된 사진

자동 생성된 설명

* 해당 노드에서 가장 큰 값을 삭제할 때는 부모 노드가 삭제할 값을 가지고 있을 것이기 때문에 부모 노드를 모두 들러 새 LargestKey로 업데이트를 해주어야 한다.

1. 삭제하려는 key가 해당 노드에서 가장 큰 값이 아닐 때

나이프이(가) 표시된 사진

자동 생성된 설명

* 이 경우에는 해당 노드에만 key 값이 존재하기 때문에 그냥 삭제한다.

1. Underflow가 발생했을 때

스크린샷이(가) 표시된 사진

자동 생성된 설명

* 만약, 1번과 2번 수행 후 Underflow가 발생되었다면 같은 레벨에 있는 형제 노드를 찾아서 Merge를 해주어야 한다. 그리고 Merge된 노드에서 새로운 LargestKey를 찾아서 모든 부모 노드를 업데이트 해주어야 한다.

1. Delete 실행



# Discussion (10 points)

- What you learned while doing your homework (contents other than class hours),  
- Describe difficulties during homework

* Btree의 remove 함수를 직접 구현해보면서 각 상황마다 어떻게 동작해야 하는지 배웠습니다.
* 하지만 insert 함수에서 첫 split 후에 파일에서 fetch를 제대로 못 해오는 현상이 발생해서 이를 계속 고치려고 해봤지만 결국 수정하지 못했습니다. 그래서 BTree 출력도 제대로 할 수 없었습니다.