**File Processing**

**Programming Project #3**

**컴퓨터공학과**

**20141602**

**황기덕**

1. **프로젝트 목적**

* 이전 프로젝트에서 구현하였던 뉴스 구독 서비스의 회원 관리를 위해 회원 정보와 뉴스 구독 내역에 관한 정보를 처리하고, 구독 내역에 대해 B – tree 자료구조를 사용하는 다단계 인덱스 (Multilevel Index)를 추가한다.

1. **프로젝트 기본 가정**

* 1차, 2차 프로젝트때 가정했던 모든 조건들을 만족한다.
* 이 프로젝트는 visual studio 2017. window sdk 버전 10.0.15063에서 진행하였다.
* 모든 인덱스가 메모리에 저장될 수 있다고 가정한다.
* NewsAgencySubscriptionSystem이 시작 될 때, 각각의 인덱스 파일의 존재 여부를 체크하여 인덱스 파일이 존재하지 않는다면 구독 리스트 파일에 대한 인덱스를 생성하도록 한다. 초기에는 레코드 파일(.dat)만 존재한다고 가정한다.
* 구독 리스트는 초기에 100개만 있고, dat file에 100개만 쓰도록 한다.
* 인덱스 key는 하나의 문자(char) 로써, 처음에는 1~100의 key를 가지고 있다. Char 범위 이상의 구독 리스트는 들어오지 않는다.
* 모든 index mode는 관리자 모드에서 실행된다.

1. **method, field 추가구현 / 추가된 파일**

* **Subscription**

Member class에서 이번 프로젝트를 위하여 추가한 field와 method이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Type** | **update** | **Get** |
| bt\_key | character | Update\_key | Get\_key |
| (static)avail\_key\_list | Char array | Init\_by\_key  Free\_avail\_key | Get\_avail\_key |

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Function** |
| makeSubscriptionIdx | Int | Subscription 의 기본 index를 만든다. |
| Bt\_searchSubscriptionIdx | Int | Index를 이용하여 dat file에서 Search한다. |
| Bt\_insertSubscriptionIdx | Int | Index file에 새로운 key, addr를 넣는다. |
| Bt\_deleteSubscriptionIdx | Int | Index file에 key를 삭제한다. |
| Bt\_modifySubscriptionIdx | int | 새로운 정보를 받아, 해당 key의 rec을 update한다. |

Bt\_key => 구독 record 각각에 하나의 key를 준다.

(static)Avail\_key\_list => static variable 로서, 현재 구독 리스트에서 어느 key가 사용되고 어느 key가 사용 가능한지 알려준다.

맨 처음 시작될 때 init\_by\_key 로 avail list를 초기화 해 주고, index list에 하나씩 넣을 때마다 get key로 가능한 키를 가져오며, delete할 때마다 free\_avail\_key로 key를 다시 사용할 수 있게끔 바꿔준다.

**추가된 파일**

생성자 / 소멸자 제외.

* **btnode**

**Field**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Function** |
| NextNode | Int | 같은 lev의 다음 node의 주소 |
| RecAddr | Int | Btree file에서 이 node의 주소 |
| MinKeys | int | Node의 최소 key 개수 |
| MaxBKeys | int | Node의 최대 key 개수 |

**Method**

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Function** |
| Pack | Int | Index/ recaddress를 pack한다. |
| Unpack | Int | Index / rec address를 unpack한다. |
| Print | void | 현재 index buffer를 출력한다. |
| Insert | Int | Node에 key와 addr을 넣는다. |
| Remove | Int | 현재 node에서 인자로 넘어온 key를 지운다. |
| LargestKey | Int | 현재 node에서 가장 큰 key를 찾는다. |
| Split | Int | 현재 node를 반으로 나눈다. |
| Merge | Int | 현재 node에 인자로 들어온 node를 합친다. |
| UpdateKey | int | 현재 node에 저장된 키를 새로운 키로 바꾼다. |

**btnode.h / btnode.hpp**

* SimplieIndex를 상속받아 제작되었다. Btree 에서 btree를 이루기 위해 필요한 node이다. Maxkeys를 사용해 최대 b node의 키 개수를 지정하고 minkey보다 높고 maxkey보다 낮은 개수로 키를 가지고 있다. Btree에서 node에 대한 method를 사용할 때, 이 class에 있는 것들을 사용한다.
* **Btree**

생성자/ 소멸자, 제외

**Field**

|  |  |  |
| --- | --- | --- |
| **Member** | **Type** | **Function** |
| Root | BTreeNode<keyType> | Btree의 root를 저장하고 있다. |
| Height | int | Tree의 height를 나타낸다 |
| Order | Int | Tree의 order를 나타낸다. |
| PoolSize | Int | 최대 tree의 height \*2를 저장한다. |
| Nodes | BTreeNode<keyType>\*\* | BTree 각 level의 node를 저장하고있다. |

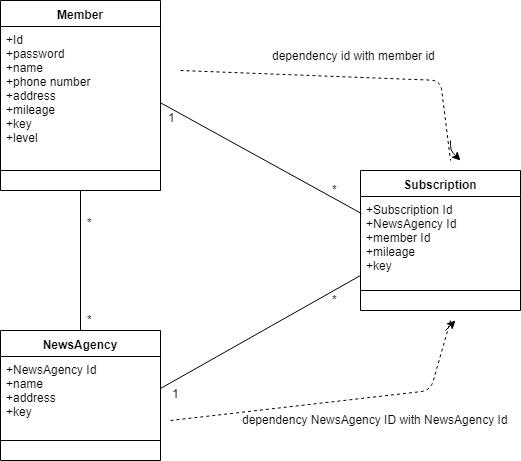
**Method**

|  |  |  |
| --- | --- | --- |
| **Method** | **Type** | **Function** |
| Open | int | 해당 파일을 open한다 |
| Create | int | 해당 파일을 create한다. |
| Close | Int | 해당 파일을 close한다. |
| Insert | Int | Btree에 해당되는 key를 넣는다 |
| Print | Void | 현재 btree를 지정된 stream에 출력한다. |
| Remove | Int | Btree에 해당되는 key를 삭제한다. |
| Search | Int | Btree index에서 해당 key를 찾아 해당 recaddr를 돌려준다. |
| FindLeaf | BTreeNode<keyType>\* | 해당 key를 가진 leaf node를 가져온다. |
| Fetch | BTreeNode<keyType>\* | 해당 addr를 가진 btnode를 가져온다. |
| Store | int | 해당 btnode를 저장해 적용한다. |

**btree.h, btree.hpp**

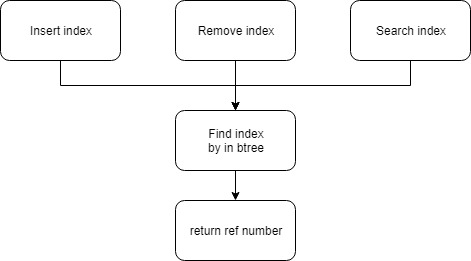
* 실제 Btree를 운영하는 함수이다. Btree 하나가 있고, 그 안에 btnode 여러가지가 tree형식으로 연결되어 있다. Insert, remove, update등의 method로 btree를 수정한다. 이 btree에 subscription list의 index가 들어가있다.

1. **클래스 다이어그램**

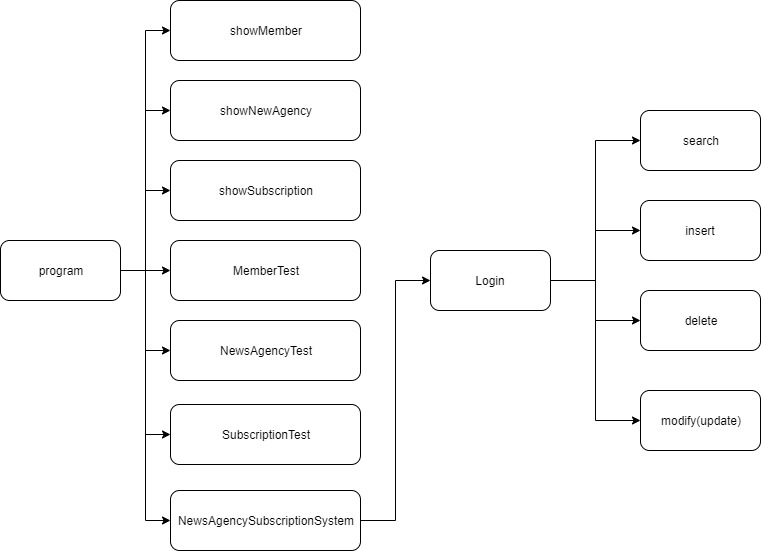


1. **Flow chart**

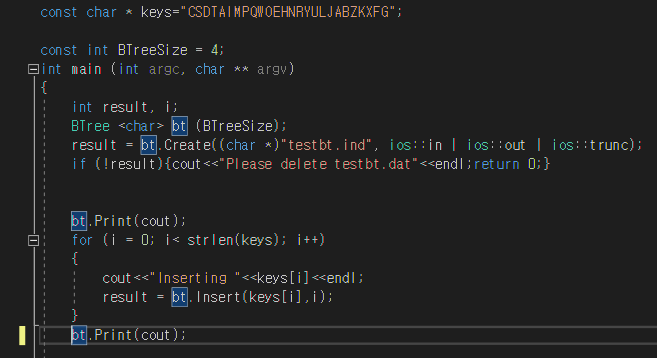
**In Subscription list, find index**



**전체 structure**

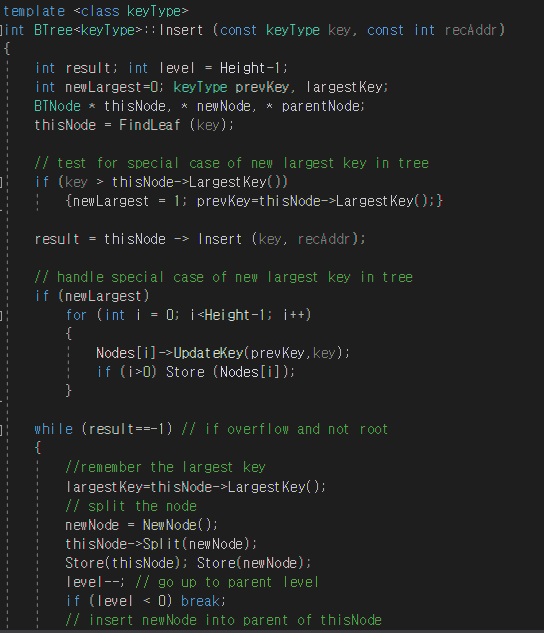


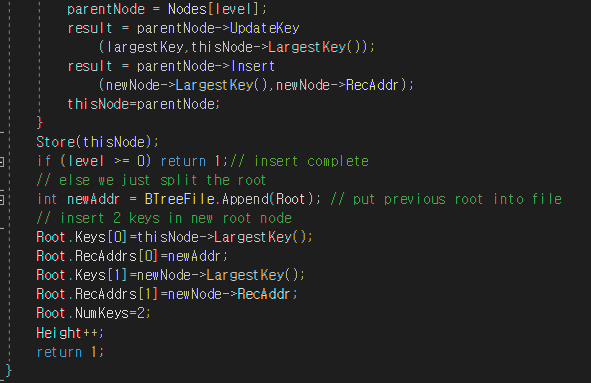
1. **보고서 요구사항**
2. **2. 프로젝트 기본 가정에 작성**
3. **4. 클래스 다이어그램에 작성**
4. **대화형 프로그램으로, 실행 후 7번을 눌러 기본 system에 들어가서 admin 계정으로 로그인을 하고 (id : admin, pw : adminpass) insert, delete, modify 등의 함수를 수행한다.**
5. **1,2 차 프로젝트에서 사용했던 subscription list에서 100개만 남기고 모두 지워버렸다. 그래서 처음엔 100개의 record만 index를 통해 구현된다. 사용자가 system에서 subscription list를 추가하거나 삭제할 수 있다.**
6. **작성한 btree, btnode의 소스코드를 subscription list에 대해서 알아보기엔 좀 힘들다고 판단해, 코드를 그대로 가져와서 알아보기 쉬운 예시 프로그램에서 돌려보았다.**



**Test code의 모습이다.**

**Insert code의 모습이다.**

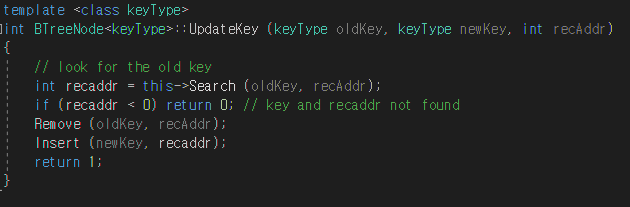




**Key를 가진 leaf node를 찾고, 만약 찾은 node의 largest key가 삽입하려는 key가 되면 위의 parent node부분도 바뀌어야 하니 newLargest이면 for문을 돌려 parent를 전부 갱신시켜준다.**

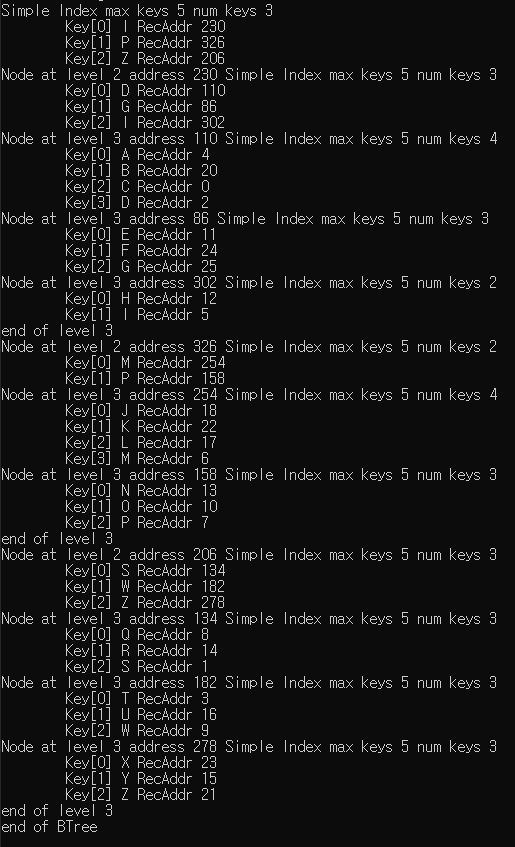
**만약 Insert할 때 overflow가 나면 split하여서 node를 두 개 나누고, parent node를 변화시켜 준다. Parent node도 overflow 될 수 있으니 while문으로 overflow가 되지 않을 때까지 반복해 준다.**

**Update 의 코드이다.**



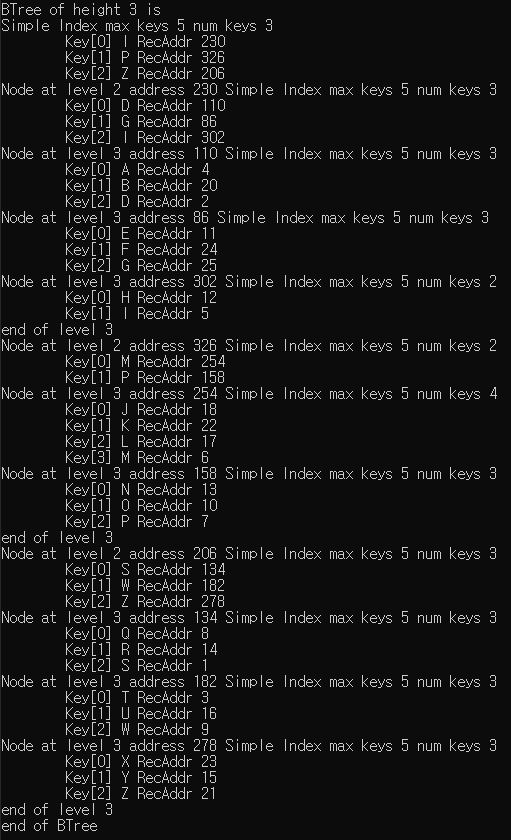
**해당 key를 remove 하고, insert하는 방법으로 손쉽게 updatekey를 해 주었다.**

**Delete는 9장 13번 문제이므로, 이 보고서 가장 마지막에 설명하겠다.**



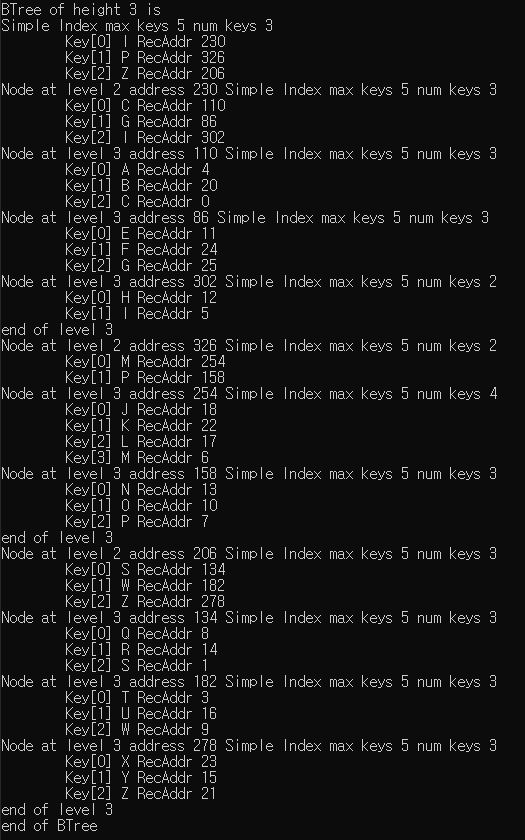
Test code의 기본 결과이다. 이 이후부터는 바뀌는 부분만 캡쳐하였다.

C 삭제시



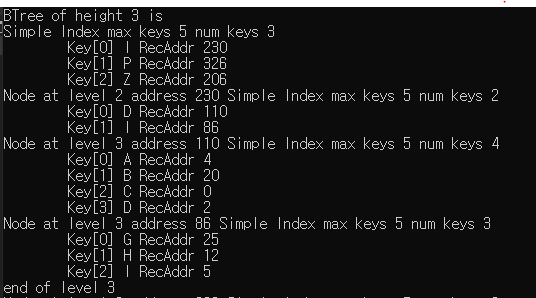
C 키가 사라졌고, node에 제대로 반영되었다.

D 삭제시



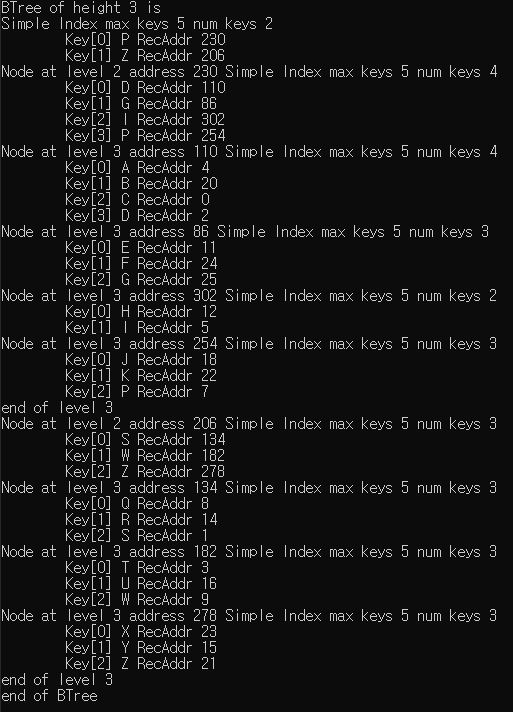
D 키가 사라졌고, D가 그 노드의 가장 큰 키였으니 제대로 반영되었다.

EF 삭제시



EF가 사라지며 G 키 하나만 남아 underflow가 되자 옆의 node와 합쳐진 모습이다. 그러면서 parent node도 제대로 반영이 되었다.

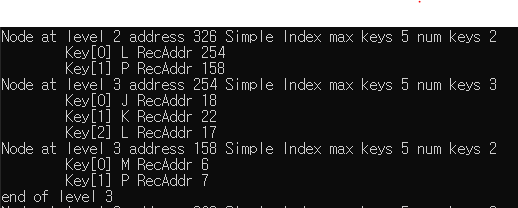
LNMO 삭제시



MP를 가지는 level 2 node, JKLM, NOP를 가지는 level3 node가 있었는데, LNMO가 삭제되며 level 3 node 두 개가 합쳐지고 level2 node가 하나만 남게 되자 level 2 node도 역시 옆의 DGI를 가지는 node와 합쳐졌다.

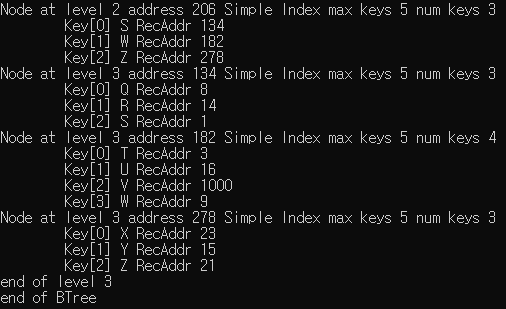
Leafnode가 변함에 따라 parent node도 underflow 되었는데, 이 역시도 잘 처리되었다.

NO 삭제시



NOP를 가지는 노드에서 NO가 사라졌지만, 옆의 node와 합쳐지면 overflow가 나는 상황이라서 옆의 node의 key를 가져와 underflow를 해결하였다.

V 삽입시



V가 삽입됨에 따라 맞는 위치의 node에 가서 삽입되었다.

1. **Btree index를 통해 데이터를 접근하는 함수는 총 5가지가 있다. Subscription.h 파일에 선언되어 있고 120줄 이후로 존재한다. 이 함수들은 subscription.cpp 파일에 474 줄 이후로 정의되어 있다.**

**먼저 bt\_makeSubscriptionIdx(subscription.cpp, 475)는 맨 처음에 btree를 이용하여 index를 만드는 함수이다. Dat 파일을 읽어서 btree를 만들고 각 key를 insert한다.**

**Bt\_searchSubscriptionIdx(subscription.cpp, 517)은 btree의 index를 이용하여 dat에서 파일을 찾는 함수이다. btree에서 key를 search해 addr를 가져와 해당되는 addr을 dat에서 읽는다.**

**Bt\_insertSubscriptionIdx(subscription.cpp 550) 은 btree에 새로운 key값을 넣는 함수이다. 이 함수는 1차와 2차 때 사용했던 insertSubscript (subscription.cpp, 252) 함수에서 id 대신 key를 받고, id 등을 입력 받아 그 rec에 key를 지정해주고 지정 받은 key와 dat 파일에 쓰여진 위치를 인자로 받아 btree에 insert해 준다.**

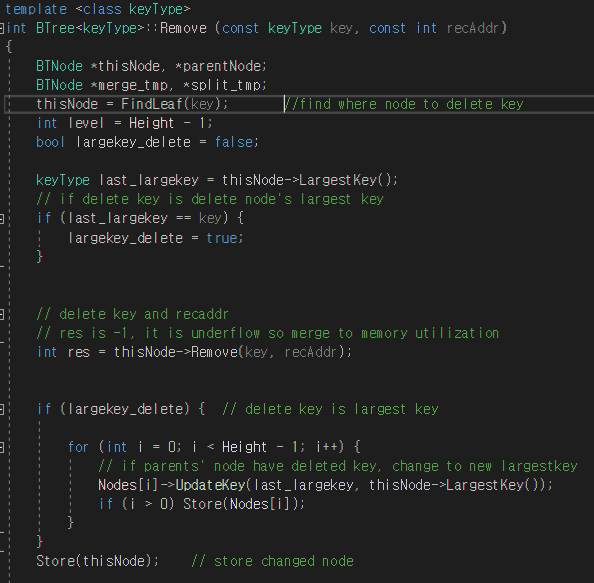
**Bt\_deleteSubscriptionIdx(subscription.cpp, 558)은 btree에 해당되는 key값을 가진 node를 삭제하는 함수로,recodrDelete(main.cpp, 371)에서 직접 호출해 주었다. 해당되는 key값을 삭제할 때, key를 재사용할 수 있게끔 key를 free해 주었다.**

**Bt\_modifySubscriptionIdx(subscription.cpp, 565)는 btree의 해당되는 key값을 찾아와서 수정하고, 다시 dat파일에 쓰고, ind 파일에 해당 key의 recaddr을 update 해 주었다.**

1. **5)번에서 같이 설명하였다. delete방식에 대한 설명은 보고서 말미에 따로 작성하였다.**
2. **보고서 말미에 따로 작성하였다.**
3. **3번에 작성하였다.**
4. **9장 연습문제**

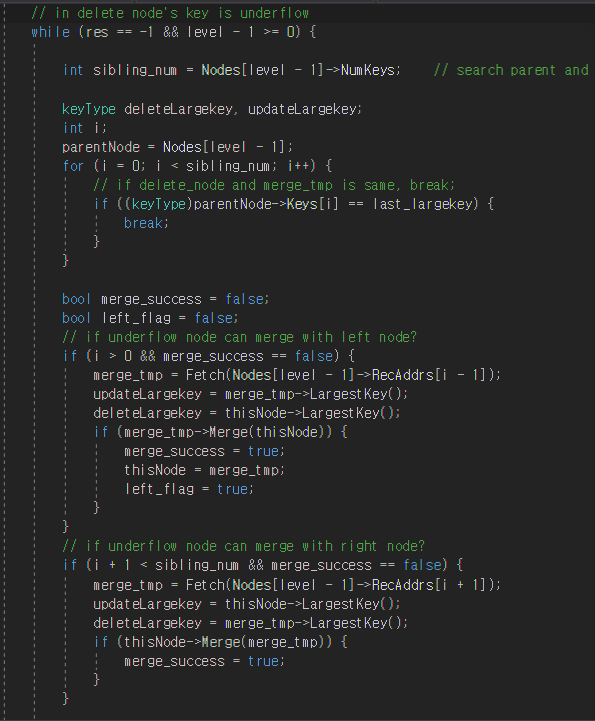
* **13**

**Implement the Delete method of class BTree**.

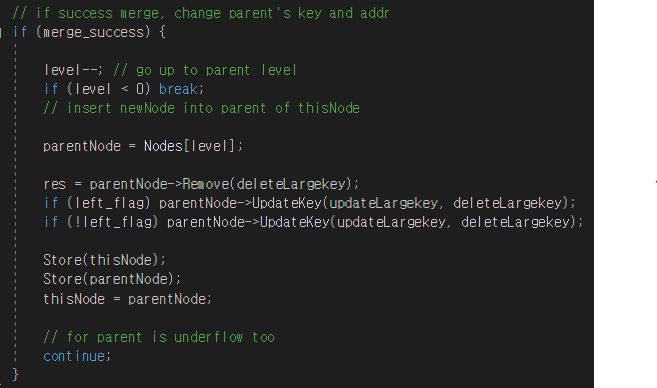


먼저 remove 함수의 초반 부분이다. Key를 가지고 있는 btree의 leaf node를 찾고, 그것을 thisNode로 지정한다. 만약 삭제하려는 key가 해당 node의 largest key이면 flag를 새워둔다.

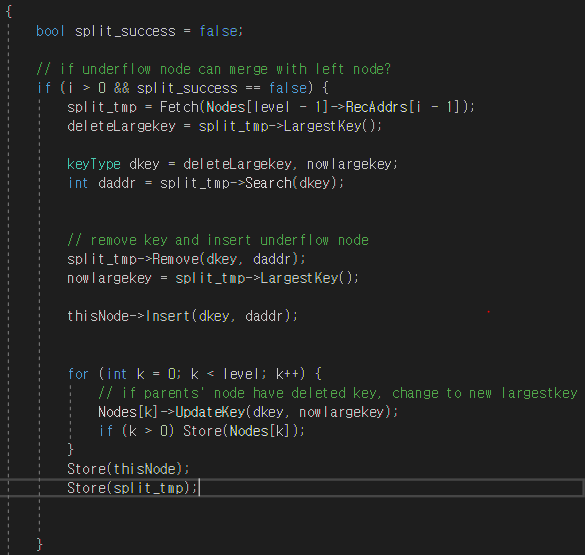
thisNode->Remove를 통해 해당 값을 지운다. 이 때 res값이 -1이면 해당 노드가underflow가 되었다는 뜻이므로 나중에 처리를 더 해주었다. 만약 지운 키가 largestkey였다면 parenet에서 더 이상 그 키를 참조해선 안되므로 위쪽의 모든 삭제된 키를 새로운 key로 바꾸어 주었다.

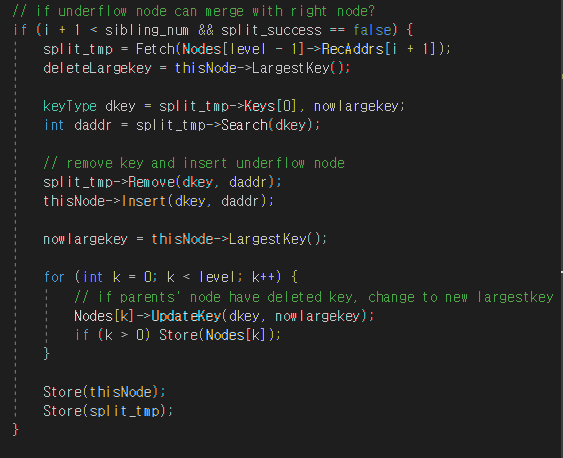


만약 underflow라면, 먼저 양 옆의 노드와 합칠 수 있는지 확인해 보았다. 먼저 현재 노드의 위치가 어느 곳인지 보고, 양 옆의 노드와 합칠 수 있는지 보았다. (if의 두 부분) 왼쪽, 혹은 오른쪽이랑 합쳐졌으면 merge\_success flag를 true로 바꾸어 주었다. 이 과정이 끝나면 underflow된 노드와 옆의 노드가 합쳐진다. 그리고 deletekey와 updatekey에 합쳐지기 전의 두 노드들의 largest key값이 저장된다.



만약 합치기에 성공했다면, parent node에서 해당 leaf node로 가는 key와 link도 바뀌어야 한다. 이전의 key값들을 지우고, 합쳐진 node의 새로운 key값과 link로 변경해 주었다. 이 작업이 끝나면 continue해서 위 과정으로 돌아가는데, 이는 parent node중 하나가 삭제되므로 parent node가 underflow 될 수도 있어서 parentNode를 thisNode로 바꾸어 다시 진행해 주었다.





만약 merge하지 못하였다면 양 옆의 node와 underflow된 node의 키를 합치면 overflow되므로 합치지 못한 것이다. 그러므로 양 옆의 node중 하나를 선택해서 하나의 key를 underflow된 node에 넣어 주었다. 이 때 parent node의 key값의 갱신만 요구하지 parent node중 key 하나가 삭제되어 underflow되는경우는 없으니 이 작업을 마치고 key만 갱신한 후 btree remove 함수를 끝냈다.