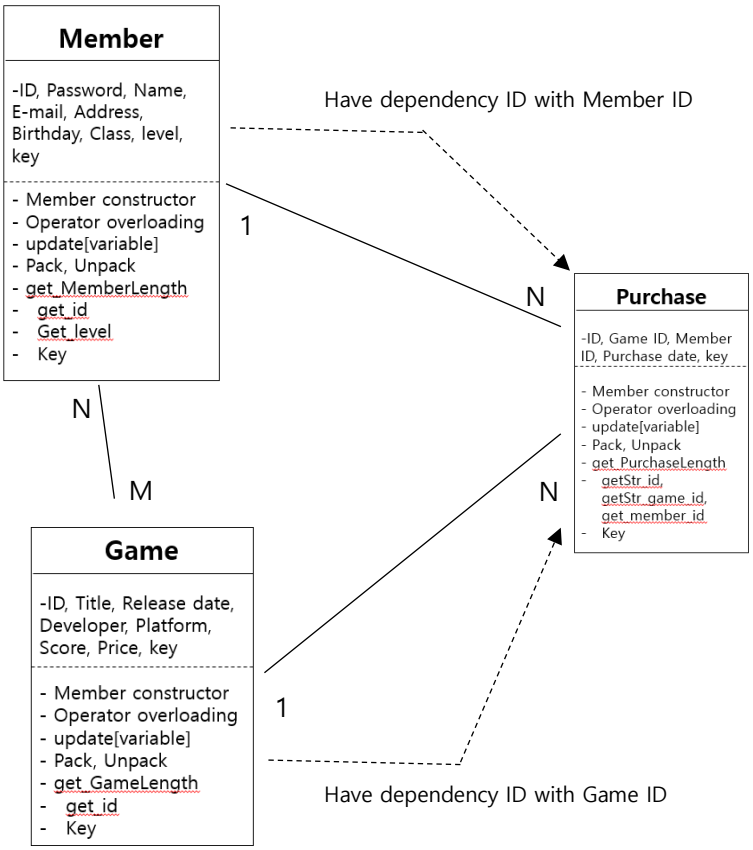
파일 처리 PR3

20161598 손동현

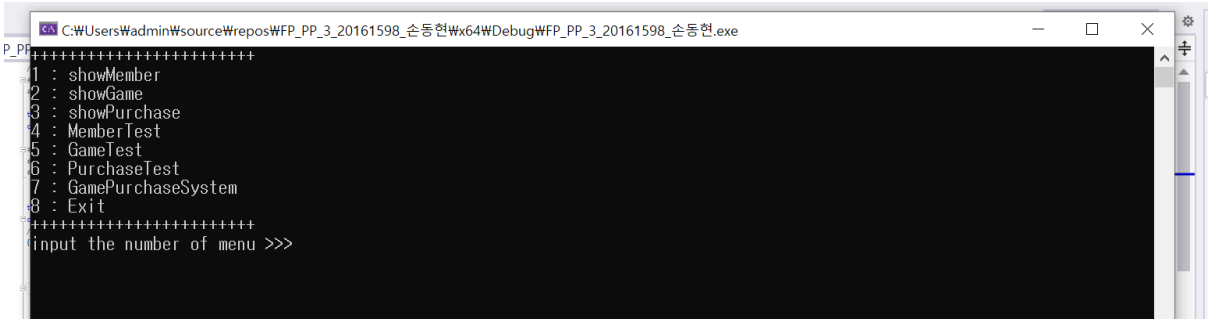
1. 프로젝트 진행 중 필요한 가정

- 가변 길이 필드들을 갖고 있는 가변 길이 레코드 구조이다.
- 레코드들의 필드들의 타입에 따라 길이를 제한할 수 있다.
- 레코드의 필드들은 구분자 '|'로 구분한다.
- 모든 경우에서 참조 무결성을 보장하여야 한다.
- GamePurchaseSystem에 들어가기전 처음 메뉴에서 4~6번을 눌러 각 레코드들에 대한 올바른 .dat 파일을 만든 후, GamePurchaseSystem을 진행한다고 가정한다.
- Member나 Ticket에 새로운 정보를 100개 이상 추가하지 않는다.
- 각 id에 대해 입력할 때, 되도록 요구조건에 맞춰 올바르게 입력한다. Key의 경우, 하나의 문자를 사용하므로, ASCII 코드 0~127번을 사용한다. 따라서 Key로 검색할 경우, 0~127 사이의 정수 값을 입력한다.

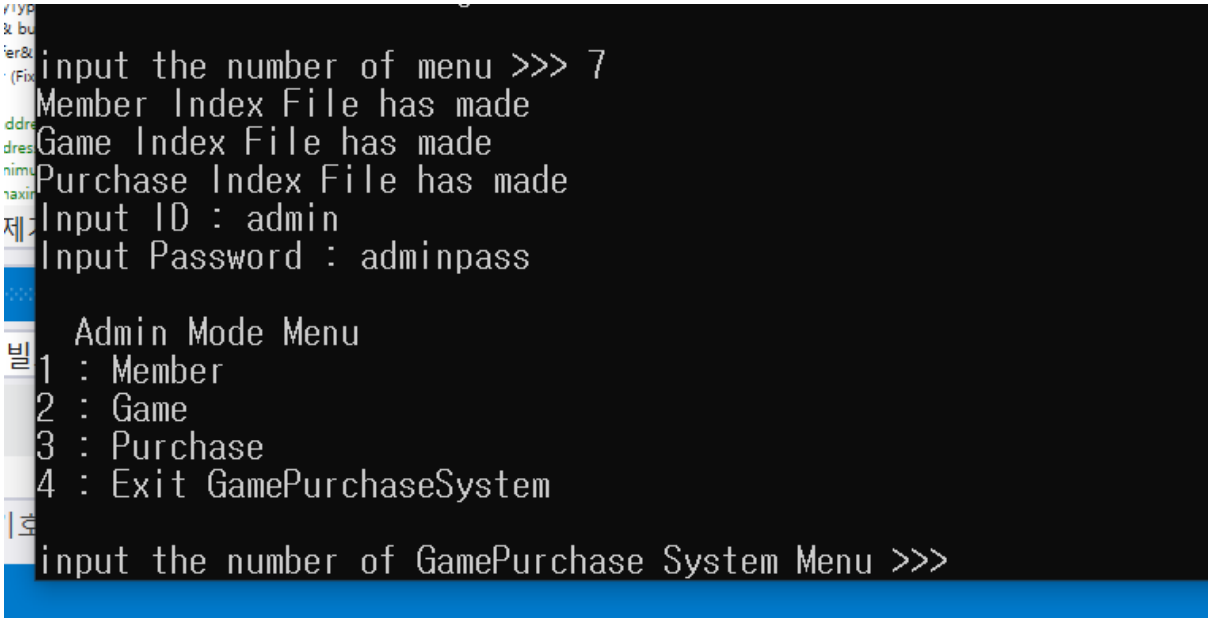
2. 구현한 클래스의 다이어그램(standard UML specification)



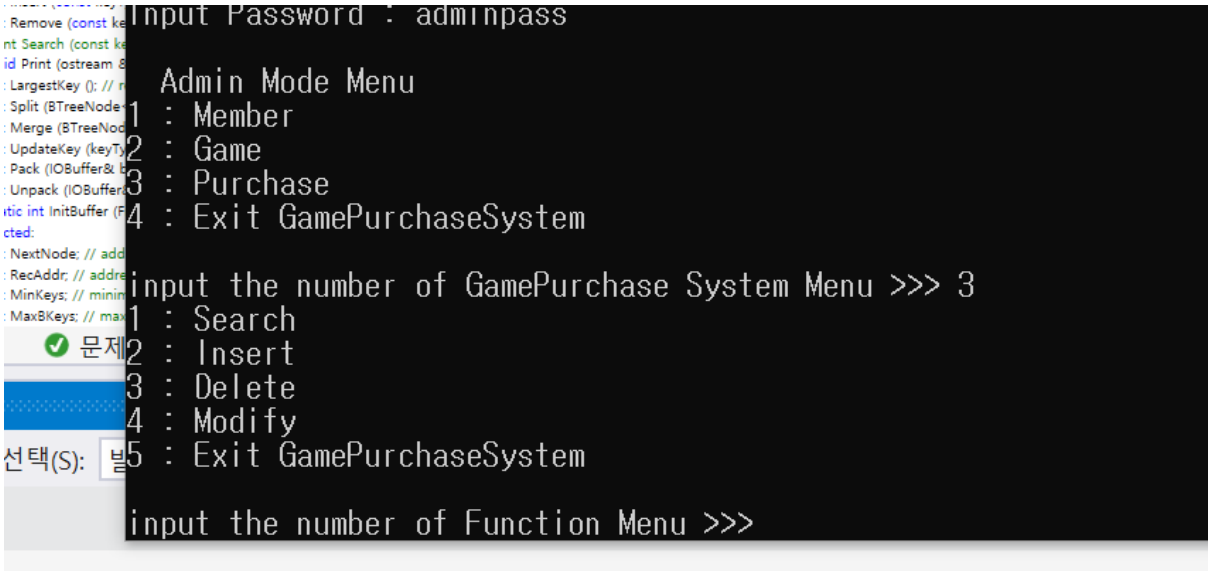
3. 프로그램 실행 화면 캡처 및 동작 방법에 대한 설명



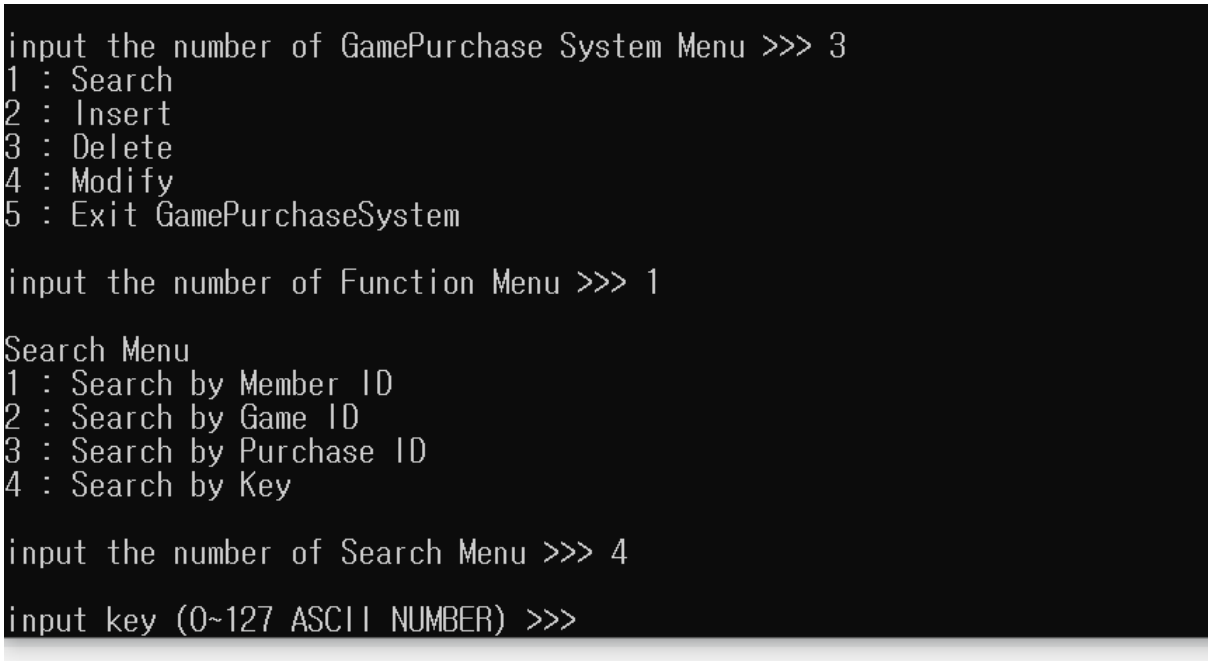
- 처음 실행하면 위와 같은 화면이 나온다. 여기서 4,5,6을 차례대로 눌러서 각 레코드에 대한 .dat 파일을 생성한다. 그 후, 7번을 눌러, GamePurchaseSystem을 시작한다.



- ID와 비밀번호를 치기 전에 index 파일이 생성되었는지 확인 후, 생성되지 않았으면, 각 레코드들에 대한 index 파일을 생성해준다. 이는 GamePurchaseSystem 실행 시 자동으로 체크한 후, 실행된다. 그 후, 관리자 모드를 구현하는 것이 과제이므로, 관리자 모드의 ID와 비밀번호를 치면 위와 같은 화면이 나온다.

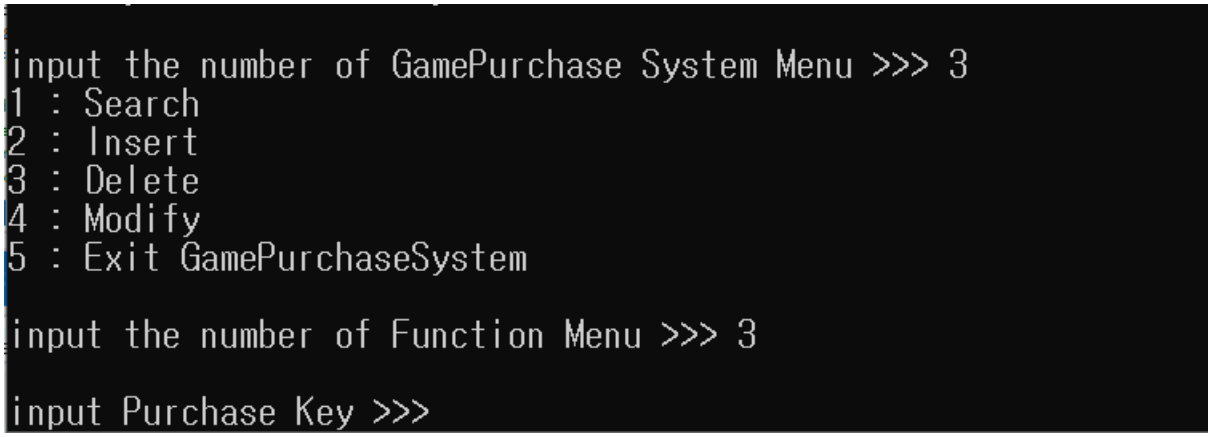


- 우리는 Purchase에 대한 구현이 목적이므로, 3번을 입력해 Purchase에 관한 메뉴가 나오게 한다.

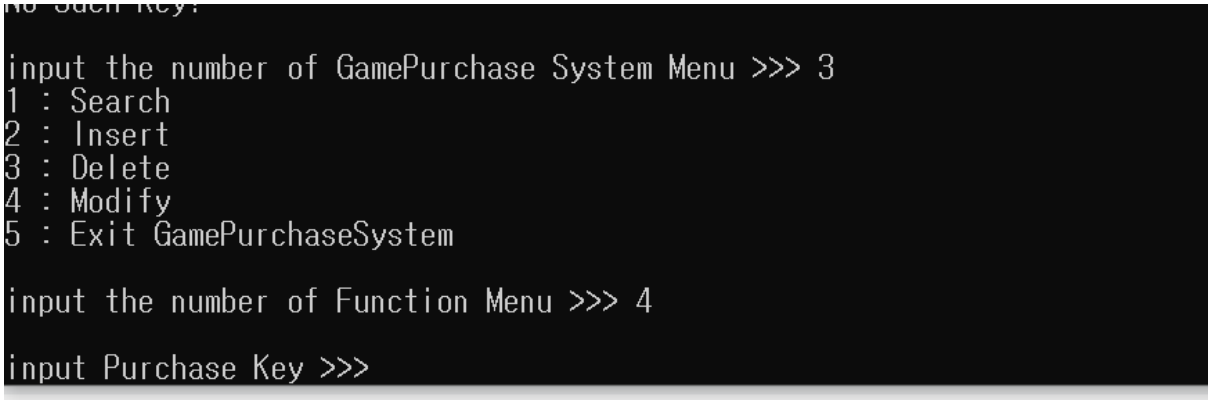


- Purchase 레코드에 대한 search 메뉴는 위와 같다. Key를 통해 찾으므로 4번을 입력해 search by Key 메뉴로 간 뒤, 0~127 사이의 정수를 입력하여 해당 key가 존재하는지 찾는다.

- Purchase 레코드에 대한 Insert 메뉴는 위와 같다. 0~127 사이의 정수를 입력하여 해당 key가 존재하는지 찾고, 존재하지 않으면, 해당 key에 대한 새로운 정보를 입력 받아 insert를 진행한다. 이 때, 0~127 사이의 정수만 입력하는 것을 주의해야 한다.

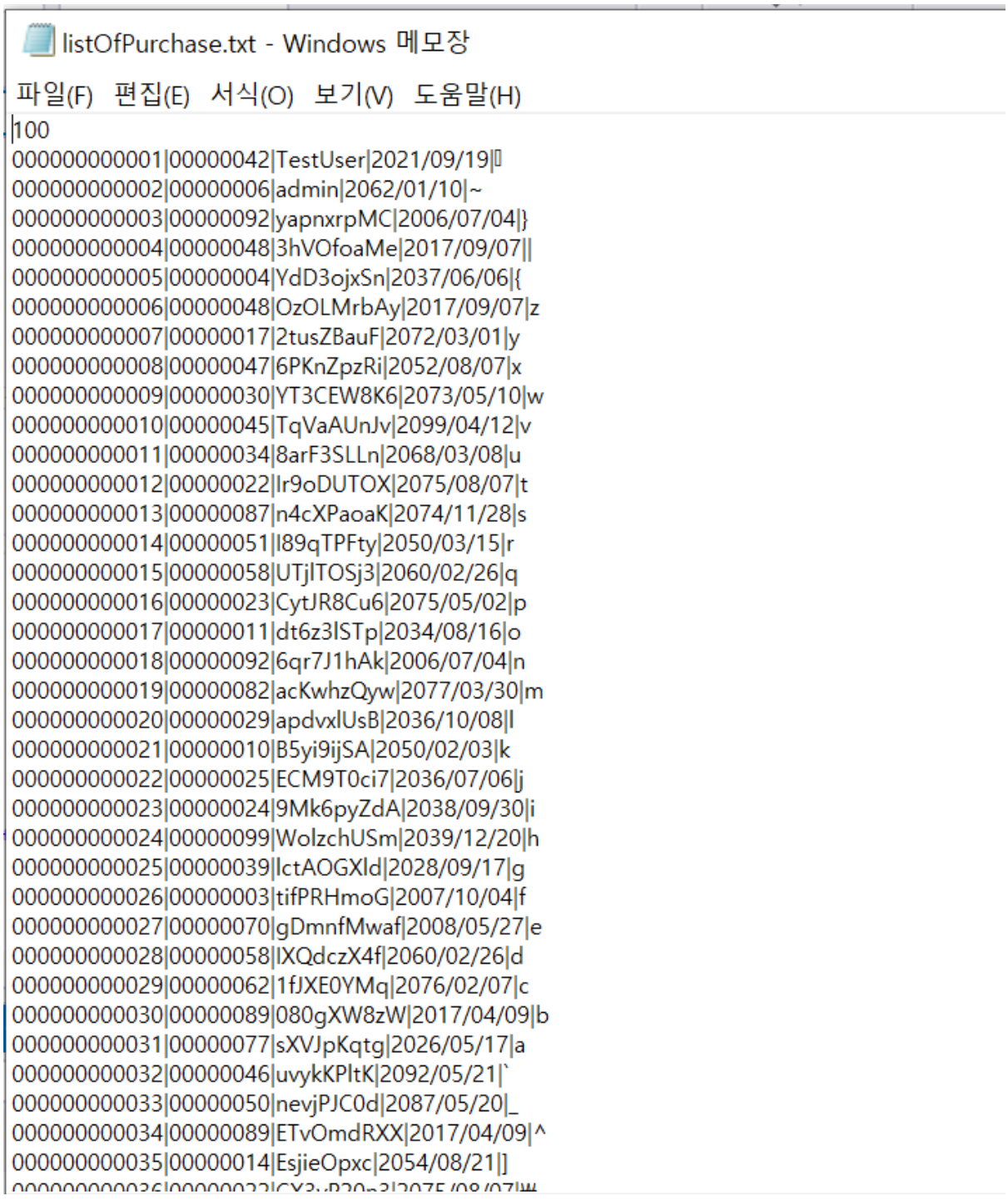


- Purchase 레코드에 대한 Delete 메뉴는 위와 같다. 0~127 사이의 정수를 입력하여 해당 key가 존재하는지 찾고, 존재하면 해당 key를 삭제한다. 이 때, 0~127 사이의 정수만 입력하는 것을 주의해야 한다.



- Purchase 레코드에 대한 modify 메뉴는 위와 같다. 0~127 사이의 정수를 입력하여 해당 key가 존재하는지 찾고, 존재하면, 해당 key에 대한 새로운 정보를 입력 받아 modify를 진행한다. 이 때, 0~127 사이의 정수만 입력하는 것을 주의해야 한다.

4. 생성한 구매 리스트에 대한 간단한 설명


listOfPurchase.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

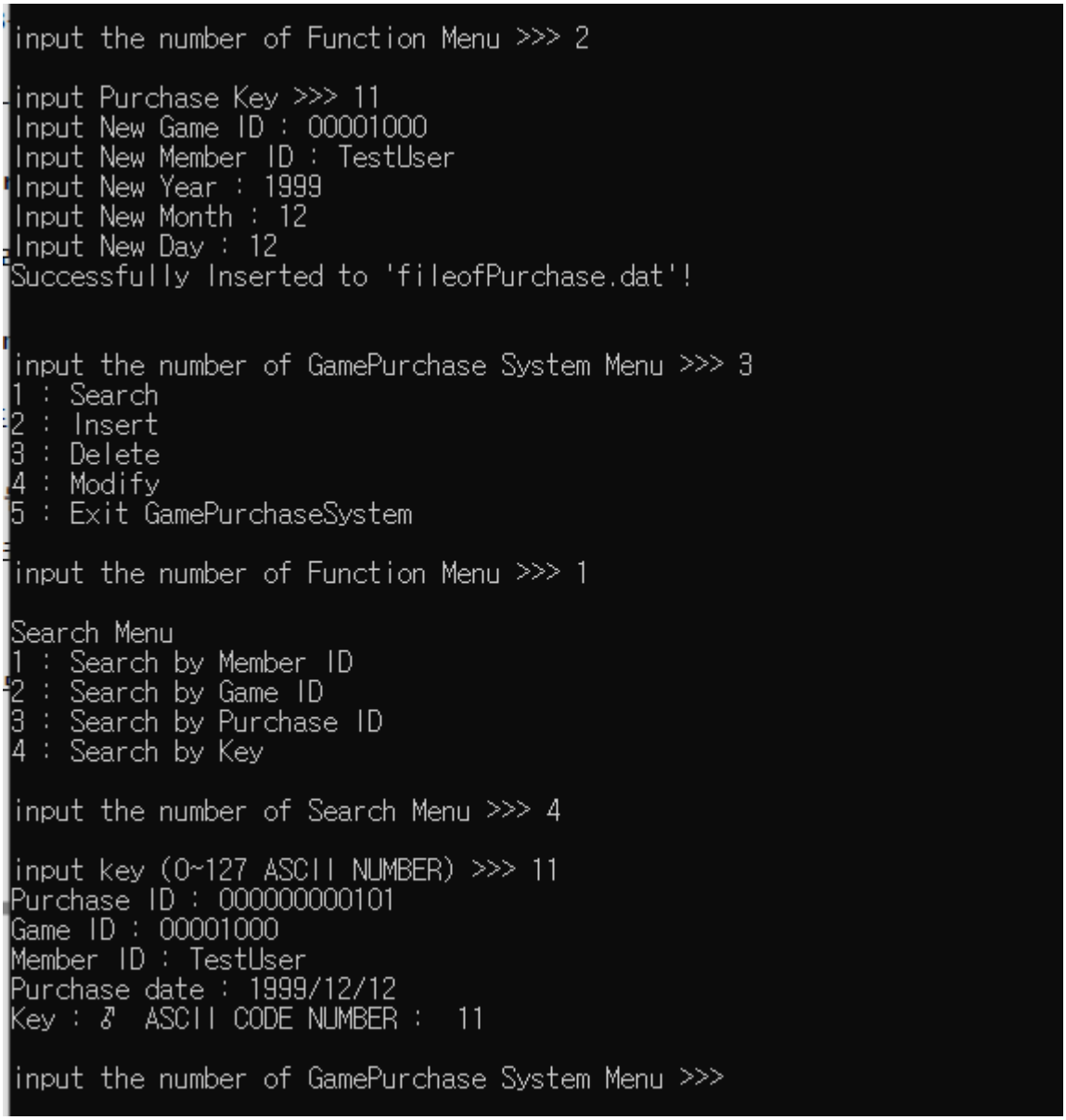
```
100
000000000001|00000042|TestUser|2021/09/19|
000000000002|00000006|admin|2062/01/10|~
000000000003|00000092|yapnxrpMC|2006/07/04|}
000000000004|00000048|3hVOfoaMe|2017/09/07||
000000000005|00000004|YdD3ojxSn|2037/06/06|{
000000000006|00000048|OzOLMrbAy|2017/09/07|z
000000000007|00000017|2tusZBauF|2072/03/01|y
000000000008|00000047|6PKnZpzRi|2052/08/07|x
000000000009|00000030|YT3CEW8K6|2073/05/10|w
000000000010|00000045|TqVaAUnJv|2099/04/12|v
000000000011|00000034|8arF3SLLn|2068/03/08|u
000000000012|00000022|Ir9oDUTOX|2075/08/07|t
000000000013|00000087|n4cXPaoaK|2074/11/28|s
000000000014|00000051|I89qTPFty|2050/03/15|r
000000000015|00000058|UTjITOSj3|2060/02/26|q
000000000016|00000023|CytJR8Cu6|2075/05/02|p
000000000017|00000011|dt6z3lSTp|2034/08/16|o
000000000018|00000092|6qr7J1hAk|2006/07/04|n
000000000019|00000082|acKwhzQyw|2077/03/30|m
000000000020|00000029|apdvxlUsB|2036/10/08|l
000000000021|00000010|B5yi9ijSA|2050/02/03|k
000000000022|00000025|ECM9T0ci7|2036/07/06|j
000000000023|00000024|9Mk6pyZdA|2038/09/30|i
000000000024|00000099|WolzchUSm|2039/12/20|h
000000000025|00000039|IctAOGXld|2028/09/17|g
000000000026|00000003|tifPRHmoG|2007/10/04|f
000000000027|00000070|gDmnfMwaf|2008/05/27|e
000000000028|00000058|IXQdczX4f|2060/02/26|d
000000000029|00000062|1fJXE0YMq|2076/02/07|c
000000000030|00000089|080gXW8zW|2017/04/09|b
000000000031|00000077|sXVJpKqtg|2026/05/17|a
000000000032|00000046|uvykKPltK|2092/05/21|`
000000000033|00000050|nevjPJC0d|2087/05/20|_
000000000034|00000089|ETvOmdRXX|2017/04/09|^
000000000035|00000014|EsjieOpxc|2054/08/21|]
```

생성한 구매리스트에 대한 정보는 위와 같다. 제일 첫 줄에 총 purchase 개수가 적혀 있고, 그 다음부터는 한 줄 당 하나의 레코드를 의미한다. 각 필드는 다음 순서를 갖고 있다. 먼저, purchase id는 12자리의 숫자, game id는 8자리의 숫자(Game Record 기록에 존재하는 Game ID 이어야 함), User ID는 string(Member Record 기록에 존재하는 Member ID이어야 함), 구매 날짜는 YYYY/MM/DD 형식을 갖고 있고,

마지막은 key로 하나의 문자를 사용하였고, 0~127 사이의 ASCII 코드 범위를 갖는다.

5. B-tree 인덱스를 생성하고 올바로 동작하는 지에 대한 검증 과정

위의 구매 리스트를 보면, 제일 위부터 key 값이 ASCII 코드 값이 127부터 1씩 감소하고 있다. 이를 통해 처음 시작했을 때, 127~28까지 index에 key 값이 저장됨을 알 수 있다.



먼저 위와 같이 ASCII CODE 11 값을 insert 한 뒤, search로 찾아보면, 정상적으로 insert 되었음을 알 수 있다.

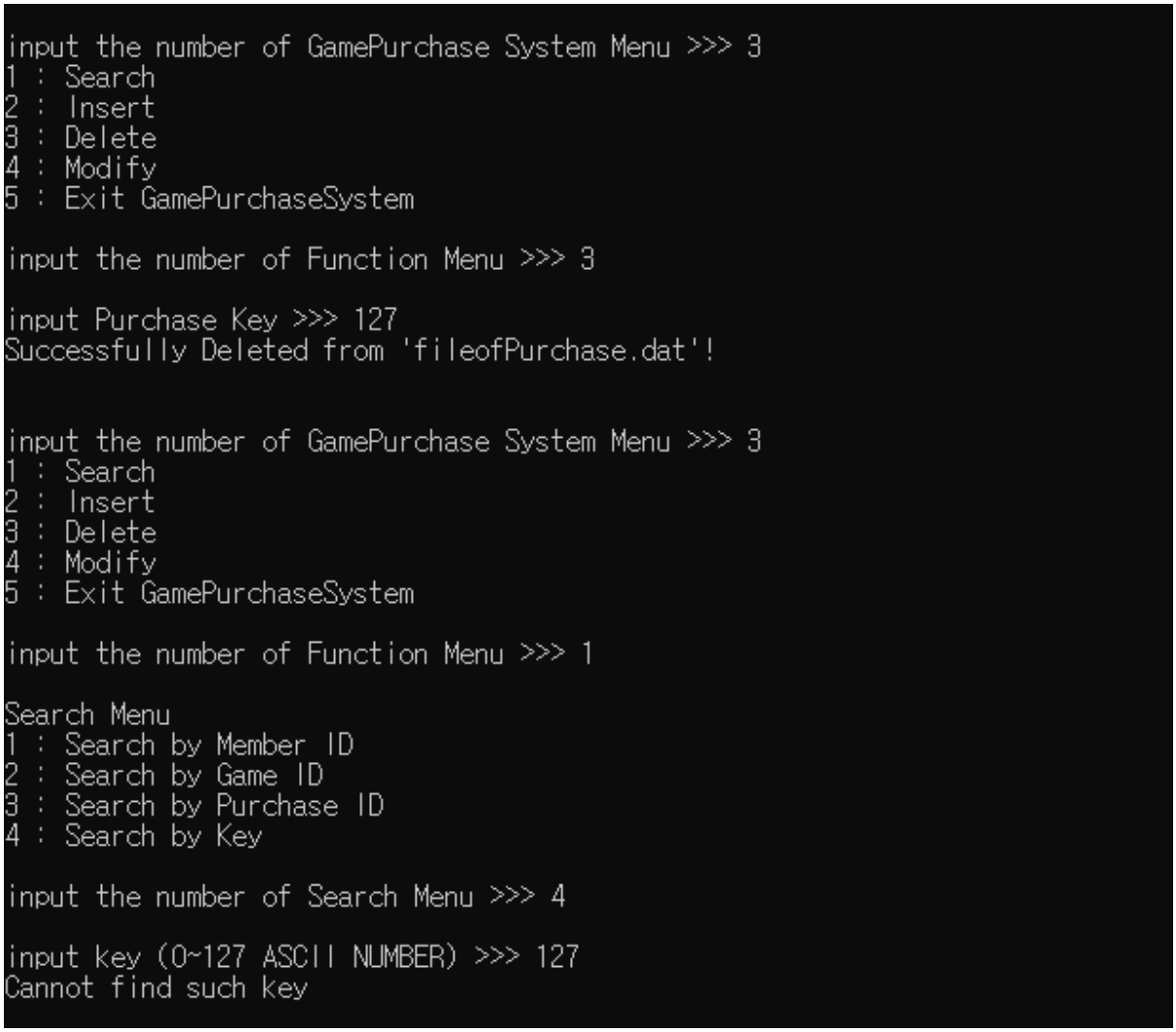
아래 코드는 insert 과정에 대한 소스코드이다.

```
1642  void insertPurchase(string IDLine, int dataNum)
1643  {
1644      //새로운 데이터 입력.
1645      Purchase insertPurchase;
1646      string line;
1647      string charstring;
1648      int inputNum;
1649      char charKey;
1650      //Admin에서 purchase insert :
1651      if (dataNum == 2)
1652      {
1653          stringstream ssInt(IDLine);
1654          ssInt >> inputNum;
1655          charKey = (char)inputNum;
1656          //if (searchPurchase(IDLine, 2, 3) != -1)
1657          if(bt.Search(charKey) != -1)
1658          {
1659              cout << "Already Exists Same Key!\n";
1660              return;
1661          }
1662          insertPurchase.update_key(charKey);
1663          //charstring = stringtoLenString(IDLine, 12);
1664          //insertPurchase.update_id(charstring.c_str());
1665      }
1666      else
1667      {
1668          while (1) {
1669          cout << "Input New Purchase ID : ";
1670              cin >> line;
1671              if (line.size() != 12)
1672                  cout << "Input 12 lengths Purchase ID : ";
1673              else if (searchPurchase(line, 3, 3) == -1)
1674                  break;
1675              else
1676                  cout << "Already Exists Same ID!\n";
1677          }
1678          charstring = stringtoLenString(line, 12);
1679          insertPurchase.update_id(charstring.c_str());
1680      }
1681      cout << "Input New Game ID : ";
1682      while (1) {
1683          cin >> line;
1684          if (line.size() != 8)
1685              cout << "Input 8 lengths Game ID : ";
1686          else if (searchGame(line, 2) != -1)
1687              break;
1688          else
1689              cout << "Input Correct Game ID : ";
1690      }
1691      insertPurchase.update_game_id(line.c_str());
1692
1693      //User에서 purchase insert
1694      if (dataNum == 1)
1695          insertPurchase.update_member_id(IDLine.c_str());
```

```
1722        cout << "Input New Month : ";
1723        while (1)
1724        {
1725            cin >> line;
1726            if (line.size() == 2 && stoi(line) < 13)
1727                break;
1728            else
1729                cout << "Input Correct Month : ";
1730        }
1731        temp += line;
1732        temp += "/";
1733
1734        cout << "Input New Day : ";
1735        while (1)
1736        {
1737            cin >> line;
1738            if (line.size() == 2 && stoi(line) < 31)
1739                break;
1740            else
1741                cout << "Input Correct Day : ";
1742        }
1743        temp += line;
1744        insertPurchase.update_date(temp.c_str());
1745
1746        purchaseCnt++;
1747        string purchaseID = to_string(purchaseCnt);
1748        insertPurchase.update_id( stringtoLenString(purchaseID, 12).c_str() );
1749
1750        char TempfileName[20] = "temp.dat";
1751        char fileName[20] = "fileofPurchase.dat";
1752        int recaddr;
1753
1754        //dat 파일의 마지막에 레코드 삽입하고 주소 값 recaddr에 갖고옴.
1755        DelimFieldBuffer buffer('|', 1000);
1756        RecordFile<Purchase>PurchaseFile(buffer);
1757        PurchaseFile.Open(fileName, ios::out);
1758        recaddr = PurchaseFile.Append(insertPurchase);
1759        PurchaseFile.Close();
1760
1761        //dat 파일의 주소 & key  b-tree에 저장
1762        bt.Insert(charKey, recaddr);
1763        //bt.Insert(insertPurchase.get_Key(), recaddr); //index 추가
1764        /* ... */
1808
1809        cout << "Successfully Inserted to '" << fileName << "'!\n" << endl;
1810        purchaseCnt++;
1811    }
```

아래 코드는 search 과정에 대한 소스 코드이다.

```
if (dataNum == 4)
{
    //key 값을 이용하여 b-tree에서 검색
    int check = bt.Search(charKey);
    //b-tree에 key 값이 존재하지 않는 경우
    if (check == -1)
        cout << "Cannot find such key" << endl;
    //if found print
    else
    {
        DelimFieldBuffer buffer('|', 1000);
        RecordFile <Purchase> RegisterFile(buffer);
        RegisterFile.Open(fileName, ios::in);
        RegisterFile.Read(p, check);
        RegisterFile.Close();
        cout << p;
        //foundP.push_back(p);
    }
}
```



위와 같이 처음에 127이 들어가 있었는데, delete를 통해 ASCII CODE 값 127을 삭제한 뒤, search로 찾아보면 해당 값이 정상적으로 delete 되었음을 알 수 있다.

아래 코드는 delete 과정에 대한 소스코드이다.

```
1813  void deletePurchase(string IDLine, int dataNum)
1814  {
1815      int inputNum;
1816      char charKey;
1817      stringstream ssInt(IDLine);
1818      ssInt >> inputNum;
1819      charKey = (char)inputNum;
1820      //if (searchPurchase(IDLine, 2, 3) != -1)
1821      if (bt.Search(charKey) == -1)
1822      {
1823          cout << "No Such Key!" << endl;
1824          return;
1825      }
1826
1827      char TempfileName[20] = "temp.dat";
1828      char fileName[20] = "fileofPurchase.dat";
1829
1830      rename(fileName, TempfileName);
1831      //temp.dat을 읽어서 fileOfMember.dat에 씀.
1832
1833      Purchase p;
1834      DelimFieldBuffer buffer('|', 1000);
1835      RecordFile<Purchase>tempFile(buffer);
1836      RecordFile<Purchase>PurchaseFile(buffer);
1837      tempFile.Open(TempfileName, ios::in);
1838      PurchaseFile.Create(fileName, ios::out | ios::trunc);
1839      int recaddr = 0;
1840      //삭제하려는 것 제외 전부 쓰기.
1841      while (1)
1842      {
1843          if (tempFile.Read(p) == -1)
1844              break;
1845          if (p.get_Key() == charKey)
1846          {
1847              bt.Remove(charKey);
1848              continue;
1849          }
1850          recaddr = PurchaseFile.Write(p);
1851          bt.Remove(p.get_Key());
1852          bt.Insert(p.get_Key(), recaddr);
1853      }
1854      tempFile.Close();
1855      PurchaseFile.Close();
1856      //bt.Remove(charKey);
1857
1858      /* ... */
1921      remove(TempfileName);
1922      cout << "Successfully Deleted from '" << fileName << "'!\n" << endl;
1923  }
```

위와 같이 아까 insert한 11 key 값에 대해 수정을 진행한 뒤, search로 찾아보면 해당 값이 정상적으로 modify 되었음을 알 수 있다.

아래 코드는 modify 과정에 대한 소스코드이다.

```
1924  void modifyPurchase(string IDLine, int dataNum)
1925  {
1926      char fileName[20] = "fileofPurchase.dat";
1927      //새로운 데이터 입력.
1928      Purchase modifyingPurchase;
1929      string line;
1930
1931      int inputNum;
1932      char charKey;
1933      stringstream ssInt(IDLine);
1934      ssInt >> inputNum;
1935      charKey = (char)inputNum;
1936      //if (searchPurchase(IDLine, 2, 3) != -1)
1937      int check = bt.Search(charKey);
1938      if (check == -1)
1939      {
1940          cout << "No Such Key!Wn";
1941          return;
1942      }
1943      //if found print
1944      else
1945      {
1946          DelimFieldBuffer buffer('|', 1000);
1947          RecordFile <Purchase> RegisterFile(buffer);
1948          RegisterFile.Open(fileName, ios::in);
1949          RegisterFile.Read(modifyingPurchase, check);
1950          RegisterFile.Close();
1951      }
1952
1953      cout << "Look at the above records and enter the purchase ID of the record you want to change >> ";
1954
1955      cout << "Input New Game ID : ";
1956      while (1) {
1957          cin >> line;
1958          if (line.size() != 8)
1959              cout << "Input 8 lengths Game ID : ";
1960          else if (searchGame(line, 2) != -1)
1961              break;
1962          else
1963              cout << "Input Correct Game ID : ";
1964      }
1965      modifyingPurchase.update_game_id(line.c_str());
1966
1967      //User에서 purchase modify
1968      if (dataNum == 1)
1969          modifyingPurchase.update_member_id(IDLine.c_str());
1970      //admin에서 purchase modify
1971      else {
1972          cout << "Input New Member ID : ";
1973          while (1) {
1974              cin >> line;
1975              if (searchMember(line, 2) != -1)
1976                  break;
1977              else
1978                  cout << "Input Correct Member ID : ";
1979          }
1980          modifyingPurchase.update_member_id(line.c_str());
```

```
1994    temp += "/";
1995
1996    cout << "Input New Month : ";
1997    while (1)
1998    {
1999        cin >> line;
2000        if (line.size() == 2 && stoi(line) < 13)
2001            break;
2002        else
2003            cout << "Input Correct Month : ";
2004    }
2005    temp += line;
2006    temp += "/";
2007
2008    cout << "Input New Day : ";
2009    while (1)
2010    {
2011        cin >> line;
2012        if (line.size() == 2 && stoi(line) < 31)
2013            break;
2014        else
2015            cout << "Input Correct Day : ";
2016    }
2017    temp += line;
2018    modifyingPurchase.update_date(temp.c_str());
2019    bt.Remove(charKey);
2020
2021    char TempfileName[20] = "temp.dat";
2022
2023    rename(fileName, TempfileName);
2024    //temp.dat을 읽어서 fileOfMember.dat에 씀.
2025
2026    Purchase p;
2027    DelimFieldBuffer buffer('|', 1000);
2028    RecordFile<Purchase>tempFile(buffer);
2029    RecordFile<Purchase>PurchaseFile(buffer);
2030    tempFile.Open(TempfileName, ios::in);
2031    PurchaseFile.Create(fileName, ios::out | ios::trunc);
2032
2033    //삭제하려는 것 제외 전부 쓰기.
2034    while (1)
2035    {
2036        if (tempFile.Read(p) == -1)
2037            break;
2038        if (p.get_Key() == charKey)
2039            continue;
2040        PurchaseFile.Write(p);
2041    }
2042    int recaddr = PurchaseFile.Append(modifyingPurchase);
2043    tempFile.Close();
2044    PurchaseFile.Close();
2045
```

6. B-tree 인덱스를 통해서 데이터 파일 레코드에 접근하는 방식에 대한 설명

- main.h의 484번 줄부터 573번 줄까지 하여서 각 레코드에 대한 index 파일을 .dat; 파일을 읽으면서 생성한다. 이 때, purchase 레코드는 저장된 주소 값과 함께 B-Tree 에 insert하여 트리를 만든다.

- 그 후, purchase 레코드의 search, delete, modify, insert 메소드들은 각각 함수를 실행할 때, B-Tree 에 접근하여, 해당 데이터에 대한 index가 B-Tree에 있는지 확인하는

방법으로 진행된다. 먼저 search 함수에 대한 코드는 main.h의 1537번 줄부터 1641번 줄로, bt.Search 함수로 해당 Key 값이 있는지 찾는다.

- Insert 함수에 대한 코드는 main.h의 1642~1811 줄로, bt.Search 함수로 해당 Key 값이 있는지 확인 후, 만일 있으면, 이미 있으므로 오류 메시지를 출력하고, 그게 아니라면 새로운 레코드에 대한 정보를 입력 받은 뒤, bt.Insert 함수를 통해 새로 저장할 Key 값과 이에 해당하는 주소 값을 저장한다. Insert 할 때는, 기존 .dat 파일의 제일 뒤에 추가하므로, .dat파일을 새로 쓸 필요는 없다.
- Delete 함수에 대한 코드는 main.h의 1813~1923 줄로, bt.Search 함수로 해당 Key 값이 있는지 확인 후, 만일 없으면, 오류 메시지를 출력하고, 그게 아니라면 해당 레코드를 bt.Remove 함수를 통해 B-Tree에서 삭제한다. 삭제를 하면, 기존 .dat 파일에 있던 다른 레코드들의 위치가 변하므로, 새롭게 .dat 만들어 써준다. 그리고 이를 실행할 때, bt.Remove 함수와 bt.Insert 함수로 기존 주소를 B-tree에서 지우고, 새로 삽입하여 준다.
- Modify 함수에 대한 코드는 main.h의 1924~2101 줄로, bt.Search 함수로 해당 Key 값이 있는지 확인 후, 만일 없으면, 오류 메시지를 출력하고, 그게 아니라면 새로운 레코드에 대한 정보를 입력 받은 뒤, 기존 레코드 index를 bt.Remove함수로 삭제하고, 새로운 정보를 bt.Insert 함수를 통해 삽입하여 수정을 완료한다. 수정을 하면, 기존 .dat 파일에 있던 다른 레코드들의 위치가 변하므로, 새롭게 .dat 만들어 써준다. 그리고 이를 실행할 때, bt.Remove 함수와 bt.Insert 함수로 기존 주소를 B-tree에서 지우고, 새로 삽입하여 준다.

7. B-tree 인덱스를 통한 삽입, 삭제 및 수정 방식에 대한 설명

- B-tree 인덱스를 통한 search 방식

btree.cpp 파일의 200번부터 205번째 줄로, search를 하는 함수로부터 key 값을 파라미터로 받아와서 이를 사용해 저장된 위치를 찾아낸다. 먼저, key값이 저장되어 있는 leaf node를 찾기 위해 FindLeaf() 메소드를 호출한다. 그리고 반환된 node를 leafNode에 저장한다. 이후에는 leafNode->Search() 메소드 호출을 호출해서 node 내에 key와 함께 저장되어 있는 .dat 파일의 주소를 찾는다. 만약 node내에 해당

key가 존재하지 않는다면 -1이 반환된다. 그렇지 않은 경우에는 key와 함께 저장되어 있는 dat 파일의 주소가 반환된다. 이 값은 다시 main 함수로 반환되어 사용자가 입력한 key에 해당하는 레코드를 .dat 파일에 접근해서 출력할 수 있도록 해준다.

- B-tree 인덱스를 통한 insert 방식

btree.cpp 파일의 64번부터 205번째 줄로, insert를 하는 함수로부터 key값과 dat 파일의 주소를 파라미터로 받아와서 인덱스에 접근한 뒤, B-tree에 삽입한다. 먼저, key 값이 저장될 leaf node를 찾기 위해 FindLeaf() 메소드를 호출하고 이로부터 반환된 node를 thisNode에 저장한다. 그리고 thisNode에서 LargestKey() 메소드를 호출해서 node내의 largest key 값을 찾는다. 만약 파라미터로 받아온 값이 이것보다 크다면 node에 largestKey라는 변수에 저장된 값을 갱신해야 한다. 따라서 newLargest 변수에 1을 저장하고 prevKey의 변수에는 node에서 현재 largest key 값을 저장한다. 이후에는 thisNode->Insert() 메소드 호출을 통해 node에 key와 dat 파일의 주소를 함께 삽입하고 이로부터 반환 된 값을 result 변수에 저장한다. 삽입 과정이 이루어진 후에는 newLargest 변수에 1이 저장된 경우에 대해 Nodes 배열을 확인하면서 prevKey의 값을 가지는 key들을 파라미터로 받아온 key 값으로 갱신한다. 이는 Nodes[i]->UpdateKey() 메소드 호출을 통해 이루어진다. 또한 result 변수에 저장된 값을 통해 삽입 과정에서 overflow가 발생하지 않았는지 확인해야 한다. overflow가 발생했다면 leaf node부터 root node까지 이동하면서 각 level 마다 key에 대한 정보를 갱신한다. 이 과정에서 삽입이 이루어지고, 다시 삽입 과정에서 overflow가 발생한다면 parent node로 이동한다. 구체적으로, overflow가 발생하면 thisNode->Split() 메소드 호출을 통해 현재의 node를 2개의 node로 나누고 새로운 node를 newNode

에 저장한다. 그리고 newNode에서 largest key값을 parentNode에 갱신해주기 위해 parentNode->UpdateKey() 메소드를 호출한다. 이후에는 newNode에서 largest key 값을 parentNode에 삽입해주면서 반환 된 값을 result 변수에 저장한다. 그리고 이 과정을 result에 -1이 저장되지 않을 때까지, 즉, 더 이상 overflow가 발생하지 않을 때까지 반복한다. while 문을 빠져나온 뒤에는 root node까지 파급 효과가 있었는지 확인해서 그 효과가 없었다면 바로 1을 반환 하여 삽입 과정이 정상적으로 이루어 졌음을 알려준다. 그렇지 않은 경우에는 root node를 split 하여 두 node의 largest key 값과 dat 파일의 해당 주소를 새로운 root에 저장한다.

- B-tree 인덱스를 통한 delete 방식

btree.cpp 파일의 119부터 197번째 줄까지 삭제를 하는 함수로부터 key값을 파라미터로 받아와서 인덱스에 접근한 뒤, B-tree에서 삭제하는 과정이다. 먼저, 삭제할 key 값이 저장되어 있는 leaf node를 찾기 위해 FindLeaf() 메소드를 호출하고 이로부터 반환된 node를 thisNode에 저장한 뒤, 이 node에서 largest key 값을 largestKey1 변수에 저장한다. 그리고 이 값이 현재 삭제하려는 key값과 일치하는지 확인하고 일치한다면 newLargest 변수에 1을 저장한다. 이후에는 thisNode->Remove() 메소드 호출을 통해 leaf node에서 현재 key 값을 삭제한다. 그 후, 아래의 3가지 경우에 나눠 진행하게 된다.

1. node에 최소 개수 이상의 key가 존재 && 삭제하려는 key 값이 largest 값이 아닌 경우 : 단순하게 삭제로 인해 갱신된 node를 저장하고 끝낸다.
2. node에 최소 개수 이상의 key가 존재 && 삭제하려는 key 값이 largest 값인 경우 : 각 level의 node에서 key에 대한 정보를 갱신해야 한다. 따라서 Nodes[i]-

>UpdateKey() 메소드 호출을 통해 삭제 전의 node에서 largest Key 값을 삭제 후의 node에서 largest key 값으로 바꿔준다. 그리고 갱신된 node들을 저장한다.

3. node에 최소 개수보다 적은 key가 존재하는 경우 : 반복문을 돌며 leafNode의 sibling node들을 찾으면서 thisNode와 merge 할 수 있는지 확인한다. merge가 가능하다면 merge를 수행하고 새로 갱신된 node를 nextNode에 저장한다. 반복문을 나온 뒤, 새로 갱싱된 노드가 thisNode의 왼쪽인지 오른쪽인지 체크한다. 왼쪽 sibling인 경우에는 largest key 값을 갱신하고, 부모 노드가 갖고 있는 node의 주소 값도 함께 갱신한다. 그리고 부모 노드로부터 largestKey2에 해당하는 key 값을 삭제한다. thisNode의 오른쪽 sibling인 경우에는 부모 노드로부터 largestKey1에 해당하는 key값을 삭제하면 끝난다.

- B-tree 인덱스를 통한 modify 방식

B-tree 인덱스를 통한 수정은 따로 함수를 구현하지 않았고, delete와 비슷하게 해당 삭제할 부분을 찾은 뒤, .dat 파일을 다시 쓰면서, 바뀐 주소들에 대해 bt.Remove() 메소드와 bt.Insert() 메소드 호출을 하여 구현된다.

8. 9장 13번 연습문제 구현

클래스 BTree의 Delete 메소드를 구현해라.

해당 문제는 btree.hpp 파일의 119번부터 197번 라인에 Remove 함수로 구현하였다.

- 먼저 FindLeaf 메소드를 호출하여 삭제하려는 key 값을 갖는 leaf node를 thisNode에 저장하고, thisNode가 갖는 largest key 값을 largestKey1 변수에 저장한다. 만일 이 값이 삭제하려는 key 값과 같은 경우, newLargest 변수를 1로 저장한다. 이후에

는 Remove 메소드에 key 값을 파라미터로 넘겨주어 B-tree에서 key 값을 삭제한다. 여기서 반환 된 값을 result에 저장하게 되는데, 만일 result가 -1이라면 underflow가 발생했음을 의미하게 된다. 만일 -1이 아닌 경우, underflow가 발생하지 않았으므로, 그 상황에서 삭제 하려는 key 값이 노드에서 largest 값이 아니라면 store 메소드를 호출하여 현재 B-tree 상태를 저장하고 Remove 메소드를 종료한다. 만일 newLargest의 변수가 1인 경우, 현재 노드의 새로운 largest key 값을 반영하기 위해 더 높은 레벨 인덱스를 변경한다. 앞서 언급한 underflow가 발생했다면, leaf node의 부모에 해당하는 Nodes[level-1]에 저장되어 잇는 node들에 대해 현재 삭제하려는 key를 포함하는 node와 합병할 수 있는지 확인한다. 합병이 가능한 경우, 합병을 하여 주고, 이 때, 합병된 sibling node가 thisNode의 왼쪽인지 오른쪽인지 확인하여 해당 노드의 parentNode에서 주소를 갱신한다. 그리고 정상적으로 삭제가 완료되면 1을 반환하게 구현하였다.

9. 프로젝트에서 사용된 소스 및 헤더 파일에 대한 설명

- Main.h에서 gamepurchasesystem을 진행하기 위한 makeindexFile 및 purchase에 관한 모든 함수들이 정의되어 있다. makeindexFile 함수는 GamePurchaseSystem을 실행할 때, index 파일을 생성하는 함수이고, PurchaseGPS 함수는 purchase에 관한 interface를 하기 위한 함수이다. insertPurchase, deletePurchase, searchPurchase, modifyPurchase 함수는 purchase 레코드에 대해 각 메소드를 행해주는 함수이다.

- btree.h / btree.hpp : B-tree class가 선언되어 있고, main.h에서 이에 대해 접근 하기 위해 사용한다.

- btnode.h / btnode.hpp : B-tree의 node가 선언되어 있고, btree 관련 파일에서 B-tree의 노드들에 접근하기 위해 사용한다.

- simpind.h / simpind.cpp : B-tree의 노드에서 인덱스에 직접 접근해 key와 recaddrs에 대한 정보를 검색, 갱신 등을 하는데 사용한다. .