**알고리즘 1차 설계과제 보고서**

**게임사 회원 관리 프로그램 설계**

**12131469김경수**

**010 8030 3206**

**wlsl4239@naver.com**

**1. 개요**

- 설계 목적 :

파일에 저장된 기존 회원정보를 입력 받고, 신규 회원가입, 회원정보 확인, 충전, 검색, 특정 회원의 최근 금액 변동내역 조회, 특정 회원의 땅 구매처리 등의 기능을 수행하며 효율적으로 회원을 관리하는 프로그램을 구현하는 것

- 요구 사항 :

1) 파일에 저장된 기존 회원정보(회원번호, 이름, 연락처, 시작좌표(x,y), 등급, 현재금액, 금액변동내역) 를 입력받아 레드블랙트리를 구축한다.

2) 질의 ‘I’ : 신규 회원가입

새로운 회원정보를 입력받아 레드블랙트리에 추가한다. 신규 회원등급과 현재금액은 0으로 시작한다. 회원번호는 유일하기 때문에, 만약 기존에 동일한 회원번호가 존재한다면, 그 노드의깊이를 출력하고, 신규 회원가입은 거절된다. 입력된 땅 좌표의 주인이 없다면, 신규 회원이 주인이 된다.

3) 질의 ‘P’ : 회원정보 확인: 특정 회원의 정보 출력

해당 회원이 존재하는지 확인하고, 존재하는 경우 회원의 정보를 출력한다. 존재하지 않는 경우, “Not found!”를 출력한다.

4) 질의 ‘A’ : 충전: 특정 회원의 캐쉬 충전처리

회원의 추가 충전된 정보를 입력받아 금액변동 내역에 추가한다. 충전처리가 완료된 후, 현재금액 변화에 따라 갱신된 회원의 등급을 출력한다. 회원이 존재하지 않는 경우, “Not found!”를 출력한다.

5) 질의 ‘F’ : 검색: 현재금액이 가장 높은 상위 5명의 정보 출력

총 현재금액이 가장 높은 상위 5명의 회원번호와 금액을, 가장 높은 금액부터 차례로 출력한다. 만약 회원이 5명보다 적으면, 존재하는 회원의 수만큼 출력한다. 현재 금액이 동일할 경우, 더 작은 회원번호의 우선순위를 높게 한다. 즉, 회원번호가 작은 회원의 정보를 먼저 출력한다. 출력할 회원이 없다면 “Not found!”를 출력한다.

6) 질의 ‘R’ : 특정 회원의 최근 금액변동내역 조회

해당 회원의 가장 최근 금액변동내역부터 차례로 변동내역을 출력한다. 만약 금액변동내역의 수가 L개보다 작으면, 존재하는 금액변동내역에 대해서만 출력한다. 금액변동내역이 없다면, “0”을 출력한다. 회원이 존재하지 않는 경우, “Not found!”를 출력한다.

7) 질의 ‘B’ : 땅 구매: 특정 회원의 땅 구매처리

회원의 구매 여부와 구매요청 후 현재금액과 땅 주인의 회원번호를 출력한다. 만약 거래금액이 땅 값보다 작거나 구매하려는 회원의 현재금액이 거래금액보다 작으면, 거래는 거절된다. 거래가 승인되면, 해당 좌표의 땅 값은 거래금액으로 갱신된다. 구매한 회원은 거래금액만큼 현재금액이 감소되고, 기존의 땅 주인은 거래금액만큼 현재금액이 증가한다. 땅 주인이 존재하지 않는 경우, 구매하는 회원의 현재금액이 감소하고 땅의 주인이 된다. 자기 자신과는 거래할 수 없고, 구매처리 후 땅의 주인이 없는 경우 D를 “–1”로 출력한다.

8) 질의 ‘Q’ : 프로그램 종료

프로그램을 종료한다.

- 프로그래밍 환경

1) OS : Windows 10, MacOS

2) 언어 : C++

3: 개발 환경 : Visual Studio 2017

**2. 필요한 자료구조 및 기능**

Pair<int,int> ground[1010][1010] : x,y좌표의 땅. 땅 주인의 회원번호와 땅값으로 구성되어있다.

Pair<int,int> Rich5[5] : 회원들 중 현재 금액이 가장 많은 5명을 저장하는 배열. 금액과 회원번호로 구성되어있다.

Index에 따라 1번째,2번째, 3번째, 4번째, 5번째로 금액이 많은 순서이다.

Node : RBT의 노드

- 멤버변수

- num : 회원 번호

- name : 회원 이름

- phoneNum : 회원 연락처

- xy : 시작좌표 x, y

- grade : 회원 등급

- nowCost : 현재 금액

- history : 금액 변동내역

- color : node의 색(0 : red, 1 : black)

- 멤버 함수

- node()

: 모든 멤버변수를 0,NULL, color = red로 초기화하고 노드를 생성한다.

- node(int \_num, string \_name, string \_phoneNum, pair<int, int> \_xy, int \_grade, int \_nowCost, bool \_color)

: 모든 멤버변수를 주어진 값으로 초기화하고 노드를 생성한다.

- void setval(int \_num, string \_name, string \_phoneNum, pair<int, int> \_xy, int \_grade, int \_nowCost, bool \_color)

: 모든 멤버변수를 주어진 값으로 설정한다.

- int depth()

: 노드의 depth를 계산하는 함수.

- void makeExternalNode()

: 노드에 external node를 생성하는 함수.

RBT : 레드블랙트리 클래스

- 멤버 변수

- root : 루트노드의 주소를 저장하는 포인터

- 멤버 함수

- RBT()

: root를 NULL로 초기화한다.

- bool doubleRed(node\* n)

: double-red가 발생했는지 확인하는 함수.

- node \* sibling(node \* n)

: sibling노드를 반환하는 함수.

- bool isBlack(node \* n)

: color가 Black인지 확인하는 함수

- bool isExternal(node \* n)

: external node인지 확인하는 함수.

- node\* search(int \_num)

: \_num을 key로 갖는 노드를 찾는 함수.

- void restructuring(node\* n)

: double-red가 발생했을 때 부모의 sibling이 black일 때 restructuring을 수행하는 함수

- node \* recoloring(node \* n)

: double-red가 발생했을 때 부모의 sibling이 red일 때 recoloring을 수행하는 함수

- void insertNode(int \_\_num, string \_\_name, string \_\_phoneNum, int \_\_x, int \_\_y, int \_\_grade, int \_\_nowCost)

: node를 red-black tree에 insert하는 함수

- void preorder(node\* n)

: 모든 node를 방문하며 rich함수를 호출하여 rich5배열을 update하는 함수.

main함수

- 파일명을 입력 받은 후 파일로부터 데이터를 읽어온다.

- 읽어온 데이터를 바탕으로 RBT를 구축한다.

- Query

- 질의 ‘I’ : 입력받은 정보로 신규 회원을 가입시킨다. 그 후 해당 회원의 depth와 승인여부(1)를 출력한다.

이미 회원이 있다면 회원의 depth와 0을 출력한다.

- 질의 ‘P’ : 입력받은 회원번호로 회원의 정보를 검색한다.

Search 함수를 수행하여 external node이면 “Not found!”를 출력하고, internal node이면 해당 노드의 정보와 depth를 출력한다.

- 질의 ‘A’ : 입력받은 회원의 현재 잔액을 충전한다.

Search 함수를 수행하여 external node이면 “Not found!”를 출력하고, internal node이면 해당 노드의 현재 잔액을 입력받은 만큼 증가시키고, 등급을 변경 후 금액 변동내역을 업데이트한다.

그 후 rich5배열을 업데이트 한다.

- 질의 ‘F’ : 현재 잔액이 가장 많은 5명을 출력한다. 회원 수가 5명 이하라면 존재하는 회원 수 만큼 출력한다.

- 질의 ‘R’ : 입력받은 회원의 최근 금액 변동내역을 입력 받은 개수만큼 최신 순으로 출력한다.

Search 함수를 수행하여 external node이면 “Not found!”를 출력하고, internal node이면 해당 노드의 최근 금액 변동내역을 출력한다. 입력 받은 개수보다 변동 내역이 적다면 변동 내역의 수 만큼 출력하고, 변동 내역이 없다면 0을 출력한다.

- 질의 ‘B’ : 입력받은 회원이 해당 좌표의 땅을 구매한다.

구매가 불가능한 경우는 두 가지인데, 첫째는 회원의 현재 금액이 거래 금액보다 적은 경우이고, 둘째는 거래 금액이 땅값보다 적은 경우이다.

구매에 실패했을 때, 0과 구매자의 현재금액, 현재 땅 주인의 회원 번호를 출력한다. 이 때 땅 주인이 없는 경우엔 -1을 출력한다.

구매에 성공한 경우 거래 금액만큼 구매자의 현재 잔액에서 차감, 판매자의 현재 잔액을 증가시킨다. 서로의 최근 금액 변동내역을 업데이트시키고 그 후 rich5배열을 업데이트 한다.

- 질의 ‘Q’ : 프로그램을 종료시킨다.

**3. 기능별 알고리즘 명세**

void rich(int cost,int num)

: 주어진 cost,num을 rich5배열에 적합한 위치에 저장하고, rich배열을 내림차순으로 유지하는 함수.

주어진 값이 rich5[0]에 저장되어야 할 경우가 최악의 경우인데, 이 경우 index 0~3에 있던 값들을 index 1~4로 이동시켜줘야 하기 때문에 이 경우에도 상수 횟수의 연산이 필요하다. 즉, 최악의 경우 O(1)의 수행시간을 갖는다.

Node

- node()

: 모든 멤버변수를 0,NULL, color = red로 초기화하고 노드를 생성한다. O(1)

- node(int \_num, string \_name, string \_phoneNum, pair<int, int> \_xy, int \_grade, int \_nowCost, bool \_color)

: 모든 멤버변수를 주어진 값으로 초기화하고 노드를 생성한다. O(1)

- void setval(int \_num, string \_name, string \_phoneNum, pair<int, int> \_xy, int \_grade, int \_nowCost, bool \_color)

: 모든 멤버변수를 주어진 값으로 설정한다. O(1)

- int depth()

: 현재 node에서 root까지 이동 후 그 length를 반환한다. red-black tree는 balanced binary tree이기 때문에 이 length는 최악의 경우 O(logN)이다. 즉 depth()는 O(logN)의 수행시간을 갖는다.

- void makeExternalNode()

: 노드에 external node를 생성하는 함수. O(1)

RBT : 레드블랙트리 클래스

- RBT()

: root를 NULL로 초기화한다. O(1)

- bool doubleRed(node\* n)

: double-red가 발생했는지 확인하는 함수. O(1)

- node \* sibling(node \* n)

: sibling노드를 반환하는 함수. O(1)

- bool isBlack(node \* n)

: color가 Black인지 확인하는 함수 O(1)

- bool isExternal(node \* n)

: external node인지 확인하는 함수. O(1)

- node\* search(int \_num)

: root부터 시작해 \_num을 key로 갖는 node를 반환한다. 최악의 경우는 external node를 반환할 경우인데 이 경우 비교 연산을 depth번 수행한다. . red-black tree는 balanced binary tree이기 때문에 depth는 O(logN)이다. 따라서 search()는 O(logN)의 수행시간을 갖는다.

- void restructuring(node\* n)

: node들의 pointer를 재설정하기만 하면 되므로 O(1)의 수행시간을 갖는다.

- node \* recoloring(node \* n)

: node들의 color를 재설정하기만 하면 되므로 O(1)의 수행시간을 갖는다.

- void insertNode(int \_\_num, string \_\_name, string \_\_phoneNum, int \_\_x, int \_\_y, int \_\_grade, int \_\_nowCost)

: node를 삽입할 위치를 찾기 위해 search()함수를 사용한다. 이 때 O(logN)의 수행시간을 갖는다.

삽입한 node가 doubleRed일 경우 restructuring 혹은 recoloring을 수행하는데, 이 중 recoloring을 수행한 후 입력받은 node의 조부모node(parent의 parent)가 doubleRed가 발생될 수 있다. 최악의 경우 이런 현상이 root까지 이어질 수 있으므로 recoloring이 O(logN)/2번 = O(logN)번 수행된다. 따라서 insertNode는 search에 O(logN), recoloring의 반복에 O(logN)으로 O(logN)+ O(logN) = O(logN)의 수행시간을 갖는다.

- void preorder(node\* n)

: root부터 시작해 모든 node를 방문하면서 rich 함수를 수행한다. rich함수는 O(1)의 수행시간을 갖고, 모든 node에서 이를 수행하므로 preorder는 O(N)의 수행시간을 갖는다.

main함수

- 파일명을 입력 받은 후 파일로부터 데이터를 읽어온다.

: 입력 파일의 회원 정보 수를 M이라고 하면 O(M)의 수행시간을 갖는다.

- 읽어온 데이터를 바탕으로 RBT를 구축한다.

: 하나의 node가 존재하는지 확인할 때 search 함수를 사용하기 때문에 O(logN)의 수행시간을 갖는다.

따라서 입력 파일에서 데이터를 읽어와 RBT를 구축하는데 O(MlogN)의 수행시간을 갖는다.

- 입력 파일로 구성한 RBT에 대해 preorder를 한 번 수행하여 현재 금액이 가장 많은 5명을 구한다. 이 때 O(N)의 수행시간을 갖는다.

- Query

- 질의 ‘I’

: 정보를 입력 받아 insertNode와 depth함수를 수행한다. 따라서 O(logN)의 수행시간을 갖는다.

- 질의 ‘P’

: 회원번호를 입력 받아 search()함수를 수행한다. 따라서 O(logN)의 수행시간을 갖는다.

- 질의 ‘A’

: 회원번호를 입력 받아 search()함수를 수행한다. 회원의 현재 금액과 등급을 변경하고 최근 금액 변동내역과 rich5배열을 업데이트한다. 이 연산들에는 O(1)의 수행시간이 필요하고, search()함수는 O(logN)의 수행시간이 필요하기 때문에 O(logN)의 수행시간을 갖는다.

- 질의 ‘F’

: rich5에 있는 내용을 출력한다. O(1)의 수행시간을 갖는다.

- 질의 ‘R’

: 회원번호를 입력 받아 search()함수를 수행한다. 최근 금액 변동내역을 주어진 개수만큼 출력한다.

이 때 O(1)의 수행시간이 필요하고, search()함수는 O(logN)의 수행시간이 필요하기 때문에 O(logN)의 수행시간을 갖는다.

- 질의 ‘B’

: 입력 받은 회원번호로 구매자를, x,y값을 통해 ground배열에 저장된 회원번호로 땅 주인을 알기 위해 search()함수를 수행한다. 땅의 거래가 끝난 후 두 회원의 현재 금액을 변경하는데 이 때 구매자가 rich5배열 내부에 있었다면 금액이 감소했기 때문에 preorder를 통해 rich5배열을 업데이트한다. 구매가 가능하고, 땅 주인이 이미 있어서 금액에 감소가 발생하는 이 경우가 최악의 경우이다. 이 경우 search() 2회, preorder 1회를 수행하기 때문에 2\*O(logN) + O(N) = O(N)의 수행시간을 갖는다.

- 질의 ‘Q’ :

: 프로그램을 종료한다.

**4. 인터페이스 및 사용법**

- 입력 파일명을 입력한다. (캡쳐 화면에서는 list\_50k.txt)

- 질의를 입력한다.

질의 ‘I’

- 질의형식: “I K N H Ax Ay”

I: 신규 회원가입 질의를 나타내는 기호

K: 회원번호

N: 회원이름

H: 연락처

Ax: 시작 x 좌표

Ay: 시작 y 좌표

- 출력형식: “D S”

D: 회원번호가 저장된 노드의 트리 내 깊이 (루트노드의 깊이는 0)

S: 회원가입의 승인 상태를 나타내는 0 또는 1인 수 (0: 거절, 1: 승인)

질의 ‘P’

- 질의형식: “P K”

P: 회원정보 확인 질의를 나타내는 기호

K: 확인하려는 회원 번호

- 출력형식: “N H G M D” 또는 “Not found!”

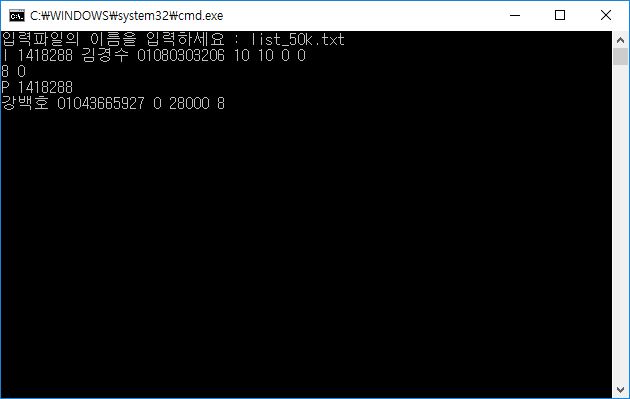
N: 회원이름

H: 연락처

G: 회원등급

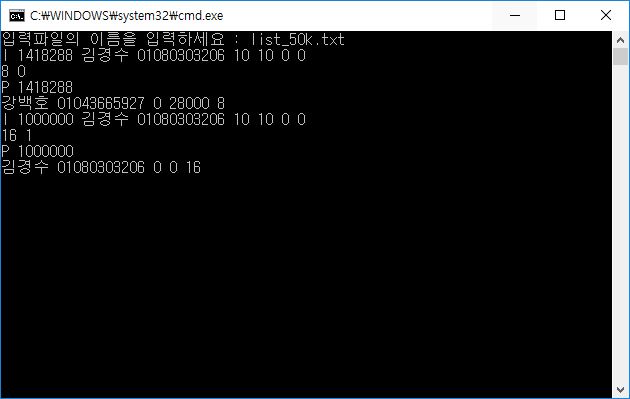
M: 회원의 현재금액

D: 해당 회원이 저장된 노드의 트리 내 깊이

****

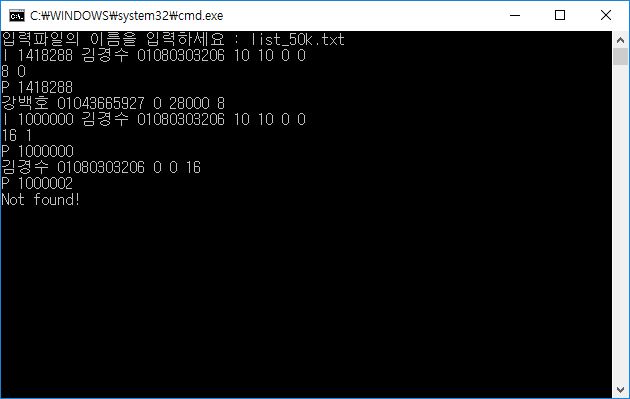
key 1418288로 “김경수”라는 신규 회원을 추가한다.

하지만 이미 1418288은 “강백호”회원의 회원번호이기 때문에 거절된다.

****

key 1000000로 “김경수”라는 신규 회원을 추가한다.

1000000은 회원이 없으므로 회원가입이 승인된다.



1000002는 해당하는 회원이 없으므로 “Not found!”를 출력한다.

질의 ‘A’

- 질의형식: “A K G”

A: 충전 처리 질의를 나타내는 기호

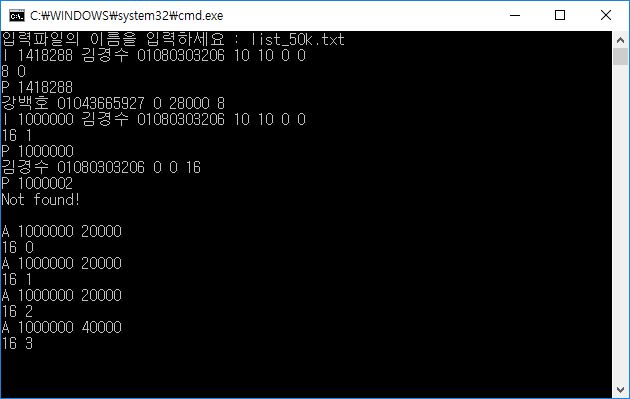
K: 회원번호

G: 충전 금액

- 출력형식: “D R” 또는 “Not found!”

D: 캐쉬 충전을 요청한 회원이 저장된 노드의 트리 내 깊이 (루트노드의 깊이는 0)

R: 충전 후 해당 회원의 등급



1000000회원에게 20000, 20000, 20000, 40000을 충전한다.

충전 후 잔액은 각 각 20000, 40000, 60000, 100000이다.

잔액이 10만 이상이면 3 / 5만 이상이라면 2 / 3만 이상이라면 1 / 3만 미만이라면 0으로 등급이 정해진다.

질의 ‘F’

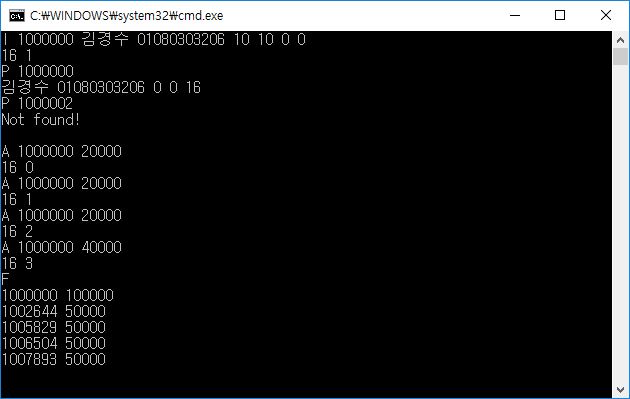
- 질의형식 : “F”

F: 검색 질의를 나타내는 기호

- 출력형식: “(조건을 만족하는 정보별로 한 줄에)P T” 또는 “Not found!”

P: 회원번호

T: 현재 금액

****

현재 금액이 가장 높은 5명을 출력한다.

질의 ‘R’

- 질의형식 : “R K L”

R: 최근 금액변동내역 조회 질의를 나타내는 기호

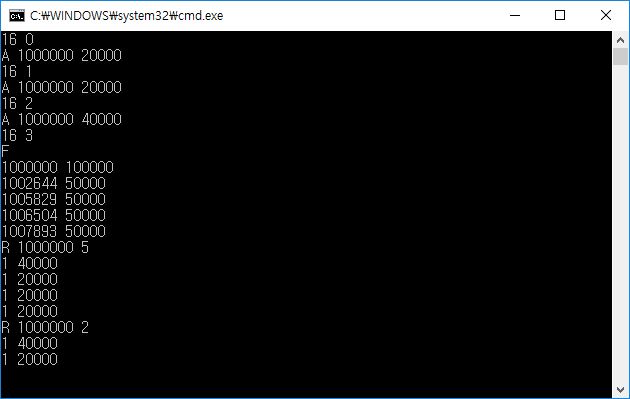
K: 회원번호

L: 조회할 최근 변동내역의 수

- 출력형식: “(조건을 만족하는 변동 금액 내역별로 한 줄에)C G” 또는 “0” 또는 “Not found!”

C: 변동된 금액의 종류 (증가 : 1, 감소 : 0)

G: 변동된 금액

****

Key가 1000000인 회원의 최신 금액 변동내역을 5개 출력한다.

위에서 질의 ‘A’를 통해 4번의 변동 내역이 있으므로 4개를 출력한다.

변동내역 2개를 출력하는 경우는 최신 변동내역부터 2개를 출력한다.

질의 ‘B’

- 질의형식 : “B N Ax Ay M”

B: 땅 구매 질의를 나타내는 기호

N: 회원번호

Ax: 땅의 x 좌표

Ay: 땅의 y 좌표

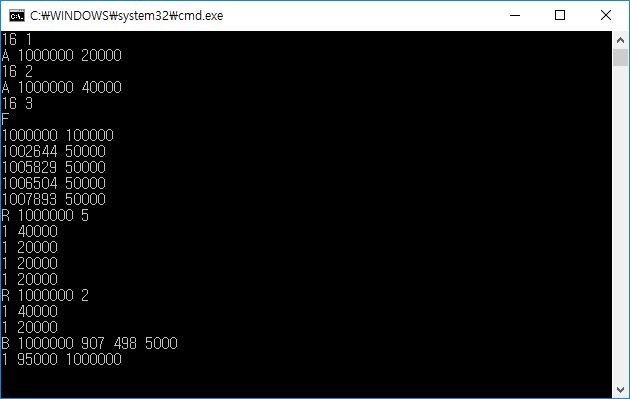
M: 거래금액

- 출력형식: “F L D”

F: 땅 구매 여부 ( 승인 : 1, 거절 : 0 )

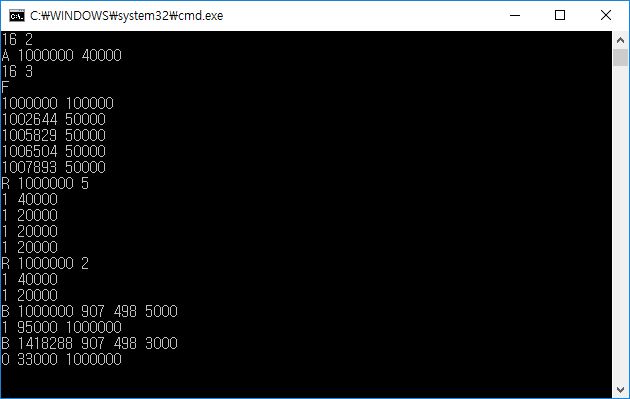
L: 구매 요청한 회원의 현재금액

D: 현재 땅 주인의 회원번호



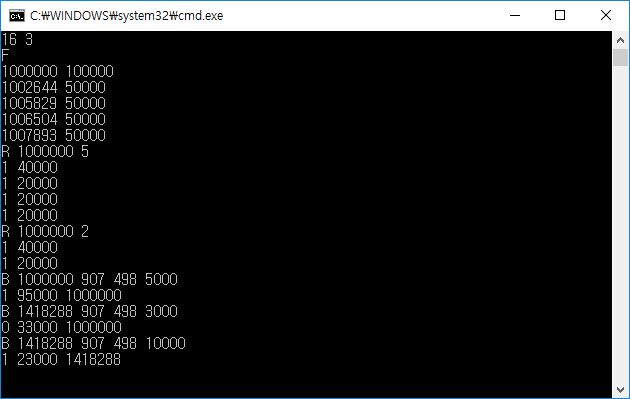
907,408땅을 key가 1000000인 사람이 5000의 비용으로 구매한다.

땅값은 이때의 거래 금액인 5000으로 설정된다.

****

907,408땅을 key가 1418288인 사람이 3000의 비용으로 구매한다.

이전 구매로 인해 땅값이 5000이므로 구매가 불가능하다.

****

907,408땅을 key가 1418288인 사람이 10000의 비용으로 구매한다.

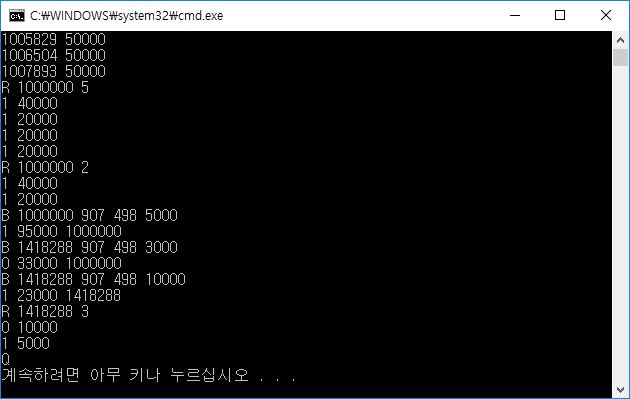
정상적으로 거래 후 땅값은 이때의 거래 금액인 10000으로 설정된다.

질의 ‘Q’

- 질의형식 : “Q”

Q: 프로그램 종료 질의를 나타내는 기호

- 출력형식: 없음



**프로그램을 종료한다.**

**5. 평가 및 개선 방향**

Red black tree는 balanced binary tree이기 때문에 삽입, 탐색을 O(logN)에 수행할 수 있다.

- 구현한 알고리즘의 장점

- 상대적으로 느린 cin,cout이 아닌 scanf와 printf를 사용했기 때문에 수행시간을 감소시킬 수 있었습니다.

- search함수를 반복문으로 구현하여 재귀함수로 구현했을 때 보다 빠른 수행시간을 갖습니다.

- 구현한 알고리즘의 단점

- 질의 B로 인해 값이 감소할 여지가 있기 때문에 이 경우 preorder를 수행해야 합니다. 이 때 rich함수를 통해 현재 cost와 num에 적합한 위치로 값을 저장하는데, 배열 size가 5라서 상수시간이 걸리긴 하지만 값을 4번 옮기므로 다른 방식을 통해 이 횟수를 줄일 수 있지 않을까 합니다.

- 향후 개선 방향

- rich5를 list로 구현하여 rich함수에서 배열 원소를 한 칸씩 이동하는 overhead를 줄일 수 있을 것 같습니다.

**6. 기타 (선택)**

- 구현 및 개발에 있어 특이 사항

- 개발 환경이 Windows의 visual studio 2017 혹은 Linux의 Eclipse CDT였기 때문에 Macbook의 XCode로 코딩하던 전 집의 데스크탑으로 과제를 수행해야 했습니다. Mac에서 코딩 후 데스크탑으로 옮기거나, 데스크탑에서만 해야 해서 효율이 떨어졌던 것 같습니다.

- restructuring을 구현할 때 옆 주석으로 4가지 구조도를 그렸는데, 이걸 보면서 구현하여 헷갈리지 않았던 것 같습니다.