

과 목 : Algorithm

과 제 명 : A theater reservation system

담 당 교 수 : 조대호 교수님

학 과 : 컴퓨터공학

Team : Team 6

이 름 : 길은규, 이유비, 최가람

제 출 일 : 2016. 06. 13.

1. Role of each member

이유비 : redblack tree 감수 및 보완, UI 구현, 프로그램 테스팅

길은규 : UI 감수 및 보완, redblack tree 구현, 프로그램 테스팅

최가람 : redblack tree, UI 감수 및 보완, 프로그램 테스팅

1. Introduction
2. Objective

 이 과제의 목적은 RBT(Red Black Tree)를 이해하고 구현해 보는 것에 큰 목적이 있다. 이진트리의 범주를 벗어나지 않으면서 자동으로 균형을 맞추는 알고리즘인 레드-블랙트리는 자료를 이진트리로 구성하여 검색 자료구조이다. 자료의 검색, 삽입, 삭제가 효율적이기 때문이다.

 이를 theater reservation system에 적용하여 구현해 봄으로 각각의 데이터가 어떻게

트리에 저장되는지 알아본다. 또한, UI(User Interface)를 구현함으로써 영화관에서 고객이 영화를 관람하고자 할 때에 어떠한 예약 절차를 통해서 영화를 예약할 수 있고, 예약 취소를 할 수 있는지를 알아 볼 수 있다.

 마지막으로 모듈별 unit testing, 전체 시스템의 integration testing을 하여, 시스템의 완성도를 높인다.

1. 시스템 동작과정(Diagram)



1. available functions

\*Main. c

|  |
| --- |
| void delay(int milliseconds) // 시간 지연 함수  void showRunningMovies() // 상영 중인 영화 목록 출력  void showScheduleOfMovie(Movie m, int date) // 해당 영화의 스케쥴 출력 함수  void reservation(int movieNum) // 예약  void cancelation(int movieNum) // 예약취소  void menu() // 메뉴를 출력해주는 함수  void showLeaves(RBT\* T, node\* nd, int\* height) // Leaf들을 출력하는 함수  void afterInsDel(int movieNum, int date, int order) // RBT의 isnertion, deletion이 이루어질 때, 노드의 숫자, root와 leaves, tree의 높이 출력. RBT의 insertion, deletion이 이루어질 떄, reservation 결과를 출력하는 함수(좌석을 보여주는 것) |

\*movie.c

|  |
| --- |
| void init\_movie(Movie\* m) // Movie 구조체를 사용하기 전에 초기화를 진행  void display\_status\_of\_seats(Movie\* m, int date, int order) // 영화의 좌석의 상태를 표시해 주는 함수 |

\*rbt.c

|  |
| --- |
| void rbtInit(RBT\* rbt) // RBT를 사용할 때, root를 null로 초기화 하고, 공통으로 사용할 nil을 넣어주는 함수  void leftRotate(RBT\* rbt, node\* x) // y가 x의 오른쪽 자식일 경우로 시작하여 left rotate 진행  void rightRotate(RBT\* rbt, node\* x) // y가 x의 왼쪽 자식일 겨우로 시작하여 right rotate 진행  void RB\_insert\_fixUp(RBT\* T, node\* z) // insert 함수에서 z노드는 red로 칠해진 상태로 넘어온다. 이 때, z->parent->color 가 또 red 라면, RBT의 4번 조건 위반. 위반하는 경우 계속 반복 ==> while(z->parent->color == COLOR\_RED)  void RB\_insert(RBT\* T, node\* z)  void RB\_delete\_fixup(RBT\* T, node\* x)  void RB\_transplant(RBT\* T, node\* u, node\* v) //RB\_transplant ( ) 함수는 v.left 나 v.right 를 update 하지 않는다. 그 부분은 이 함수의 호출자가 책임진다. 현재 u와 u의 부모와의 관계를 v와의 관계로 바꾼다. v의 부모가 u의 부모가 되고, u의 부모의 자식은 더이상 u가 아니라 v가 된다. v는 nil\_node 일 수 있다.  void RB\_transplant(RBT\* T, node\* u, node\* v)  void RB\_delete(RBT\* T, node\* z)  void RB\_free\_all\_nodes(node\* nd) // 할당해준 모든 노드를 반환하기 위한 함수 |

\*movie.h

|  |
| --- |
| void init\_movie(Movie\* m); // movie 구조체 초기화 함수  void display\_status\_of\_seats(Movie\* m, int date, int order); // 영화좌석 display 함수 해당 영화의 구조체 주소, 날짜, 그 날의 그 영화에 관해서 몇번째 영화인지를 나타내는 변수 세 개를 넘긴다. 그리고 그 영화의 좌석 현황을 시각적으로 보여준다.  void movie\_free(Movie\* m); // movie 구조체 내의 RB-tree에 대한 free 함수void all\_free(Movie\* mpp); // 랜덤하게 생성된 총 영화의 해체를 담당하는  함수, 내부에서 movie\_free() 함수를 호출한다. |

1. User interface and modules description

(including inputs and outputs)

1. 콘솔 UI + 입출력 설명



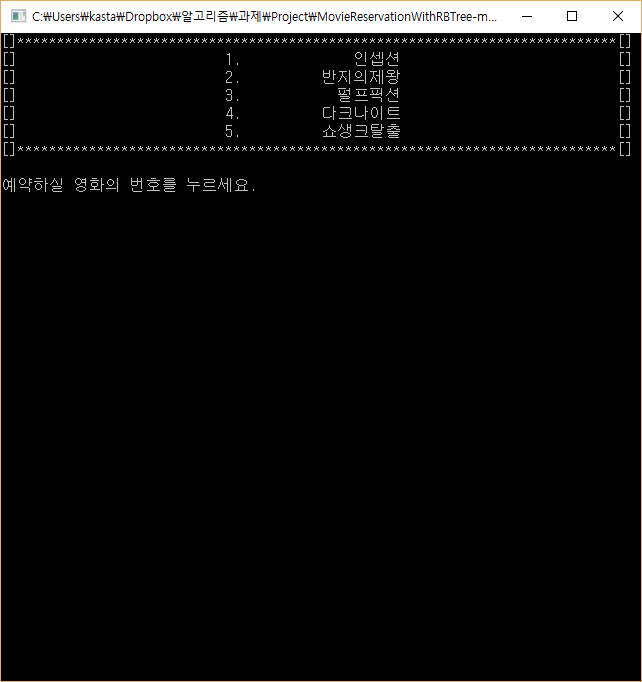
<그림 1> 처음 프로그램 실행 시 메뉴 출력 화면

1. 메뉴 출력 화면

: 프로그램 실행 시 첫 화면이다. 우선 프로그램의 출력 내용들을 좀 더 깔끔하게 표현하기 위해 콘솔창의 크기를 조금 조정했다. 이 크기는 프로그램 실행 시부터 종료 시까지 유지되며 한번에 출력되는 모든 내용들을 크기를 변경하지 않고 담을 수 있다. 또한 다음단계로 넘어가거나 처음 실행 화면으로 되돌아 가는 경우 이전에 출력되었던 내용들을 지우고 새로 출력되는 내용들만 담을 수 있도록 하여 사용자가 보기 편하도록 구현하였다.

첫 화면에서는 영화관에 온 것을 환영하는 글이 출력되고 다섯 개의 선택 가능한 영화들이 출력된다. 사용자는 영화를 선택하기에 앞서 출력된 메뉴들 중 원하는 메뉴를 먼저 선택한다. 선택하려면 해당하는 메뉴 앞의 번호를 누르면 된다. 이때 사용자가 번호를 선택 한 후 엔터 키를 눌러서 다음단계로 넘어가는 대신 번호를 누르자마자 그 번호를 버퍼에 저장하지 않고 다음단계로 넘어갈 수 있도록 getch()함수를 사용했다. 이는 나중에 설명될 좌석 선택 과정을 제외 하고 모든 경우에 적용된다. 1번이나 2번을 누르면 다음 단계로 진행되고 3번을 누르면 프로그램이 즉시 종료된다. 1~3 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.

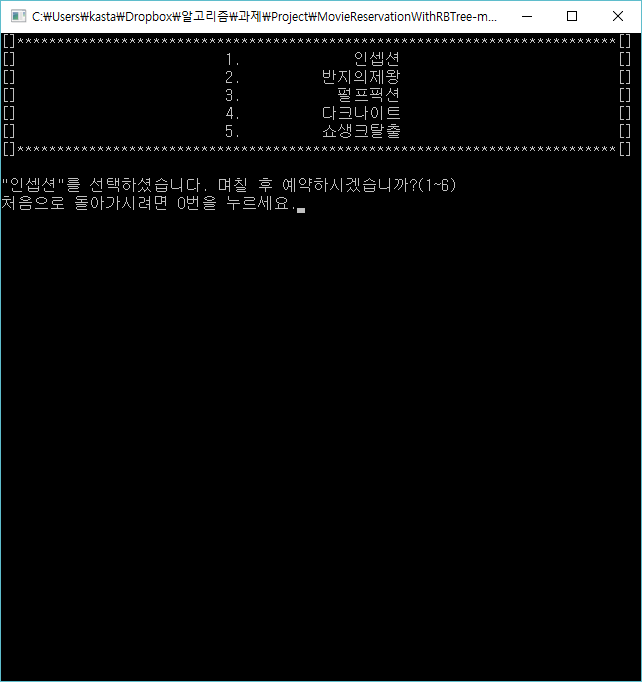
1. 예약



<그림 2> 예약 메뉴 선택 시 출력 화면

- 예약 영화 선택

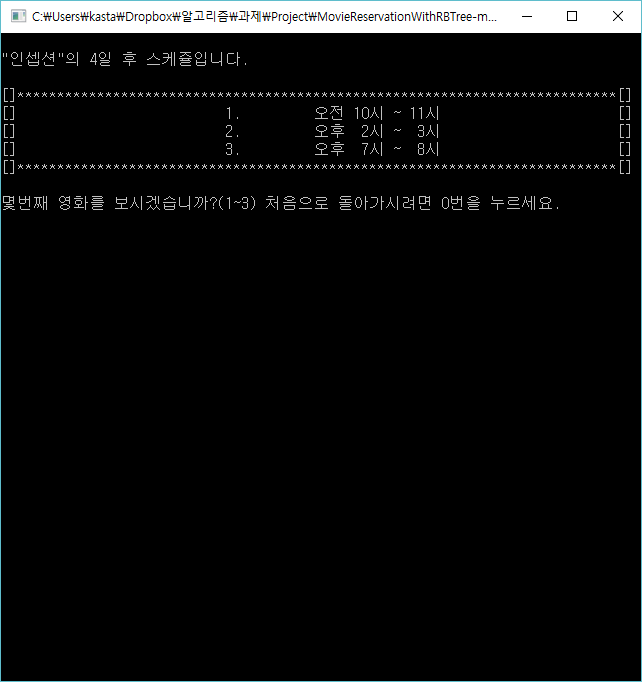
: 첫 화면에서 예약 메뉴 선택 시 출력된 영화들 중 예약하고 싶은 영화를 선택하도록 했다. 예약을 원하는 영화의 번호를 누르면 다음단계로 진행되고 1~5 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 3> 예약할 영화 선택 시 출력 화면

- 예약 날짜 선택

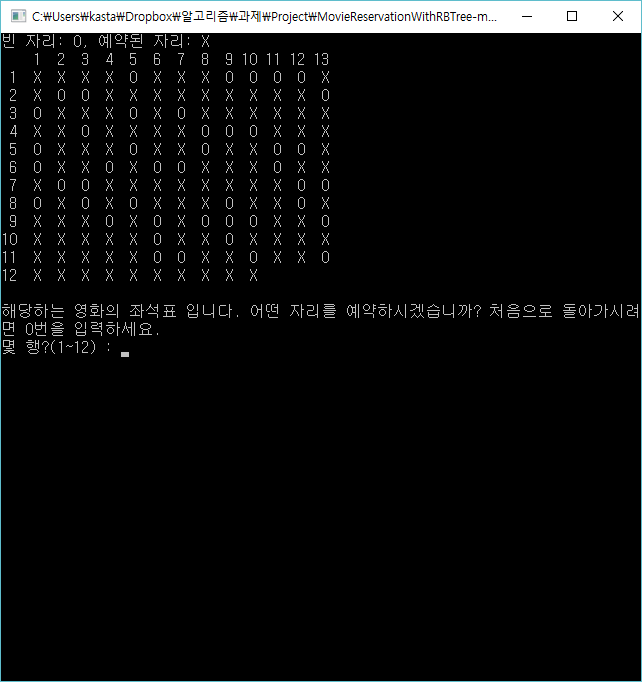
: 영화를 선택하면 그 다음으로 예약하고 싶은 날을 입력 받는다. 예를 들어 1번 영화를 선택하면 선택한 영화의 이름과 함께 며칠 후에 상영되는 영화에 예약을 하고 싶은지 묻는다. 0번을 누르면 처음 프로그램 실행 시 메뉴 출력 화면으로 되돌아간다. 0~6 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 4> 예약 날짜 선택 시 출력 화면

- 예약할 영화 시간대 선택

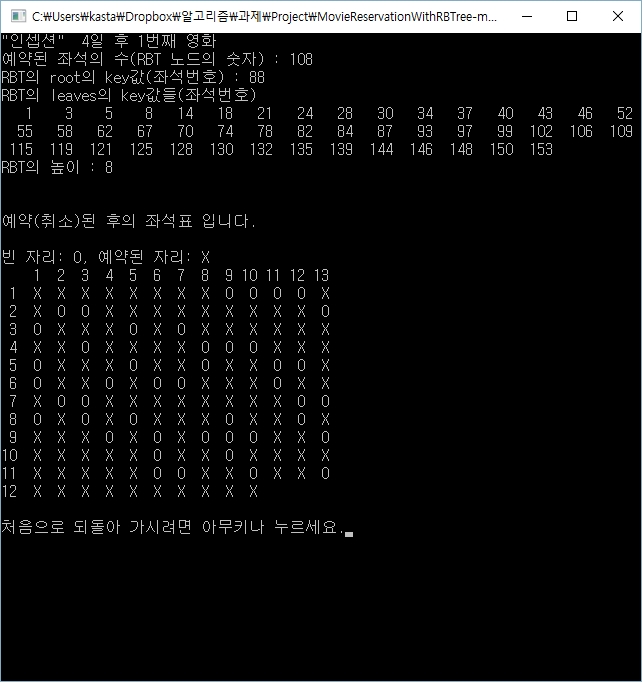
: 예약하고 싶은 날을 선택 하면 그 날의 영화 상영 시간들이 출력된다. 4일후의 영화를 예약한다고 가정하면 그 영화의 4일후의 상영 시간들이 출력되고 어느 시간대를 예약하고 싶은지 묻는다. 0번을 누르면 처음 프로그램 실행 시 화면으로 되돌아 간다. 0~3 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 5> 예약 시간 선택 시 출력 화면

- 좌석 선택

: 예약하고 싶은 시간대를 선택하면 해당 영화의 선택된 상영 시간의 좌석표가 출력된다. 사용자는 이를 보고 예약하고 싶은 좌석을 선택하고 입력하면 된다. 이때 좌석의 행과 열이 항상 두 자리 수이기 때문에 getch()함수를 사용할 수 가 없어 엔터 키를 입력하면 다음 단계로 진행되도록 했다. 만약 입력 받은 행이나 열이 출력된 좌석표의 행과 열의 범위를 초과하면 잘못된 입력임을 알리고 다시 행과 열을 입력 받도록 했다. 또한 입력 받은 행과 열에 해당하는 좌석이 이미 예약된 좌석이면 이미 예약된 좌석임을 알리고 다시 행과 열을 입력 받도록 했다. 올바른 행과 열을 입력(빈 자리)하게 되면 예약이 성공했음을 알리고 다음단계로 진행된다. 이때 잠시 동안 프로그램의 진행을 지연시켜서 사용자가 예약이 성공했음을 충분히 인지하도록 했다.

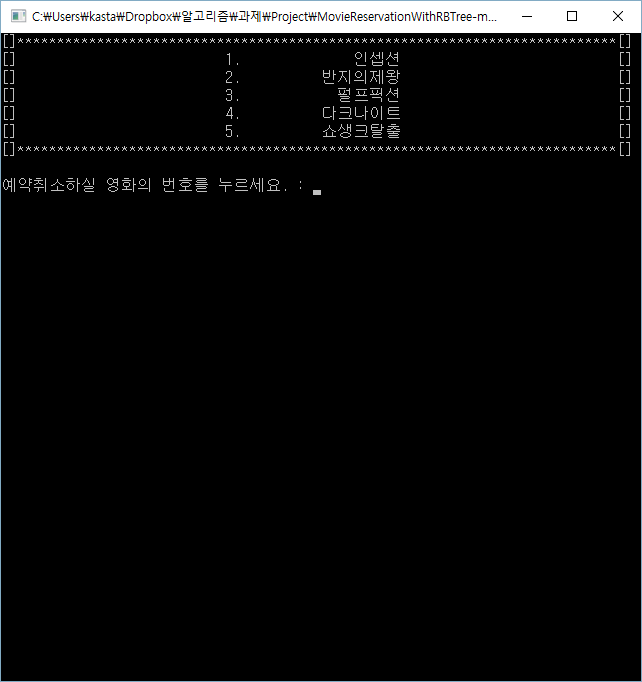


<그림 6> 예약 성공 시 출력 화면

- 예약 성공

: 예약 과정이 성공적으로 진행되면 <그림 6>과 같은 화면이 출력된다. 어느 영화를 언제, 어느 시간대에 예약했는지 사용자에게 알려주고 그 영화의 RB Tree의 node수와 root의 key값, leaf들, 높이를 순서대로 출력하도록 했다. 마지막으로 예약된 후의 좌석표를 출력하여 사용자가 원하는 자리에 성공적으로 예약이 되었음을 알릴 수 있도록 했다. 그리고 좌석표를 충분히 확인 할 수 있도록 다른 입력을 받기 전까지 다음단계로 진행되지 않도록 했다. 충분히 확인 한 후에 아무 키나 입력하면 처음 프로그램 실행화면으로 되돌아 간다.

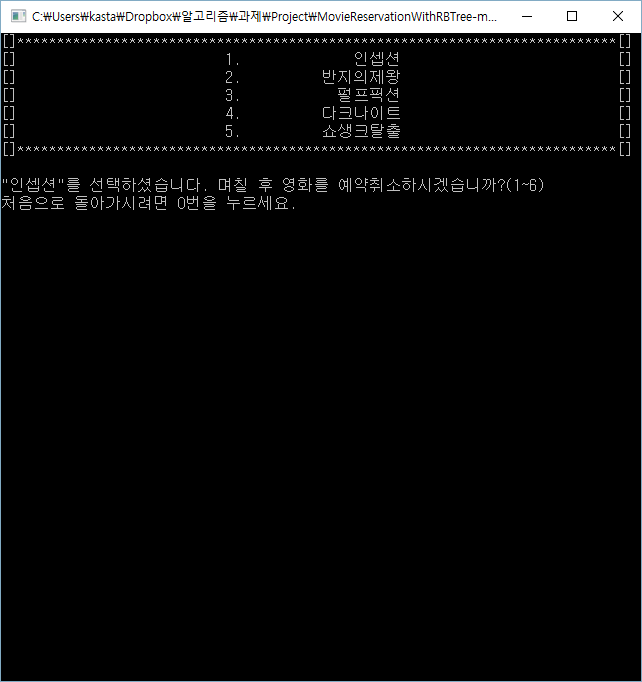
1. 예약 취소



<그림 7> 예약 취소 메뉴 선택 시 출력 화면

- 예약을 취소할 영화 선택

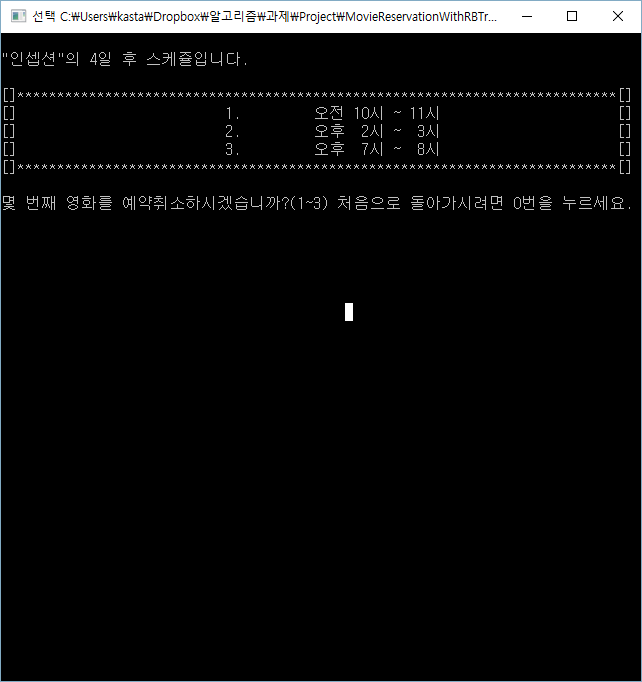
: 첫 화면에서 예약 취소 메뉴 선택 시 출력된 영화들 중 예약을 취소하고 싶은 영화를 선택하도록 했다. 예약 취소를 원하는 영화의 번호를 누르면 다음단계로 진행되고 1~5 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 8> 예약을 취소할 영화 선택 시 출력 화면

- 해당 영화의 예약을 취소할 날짜 선택

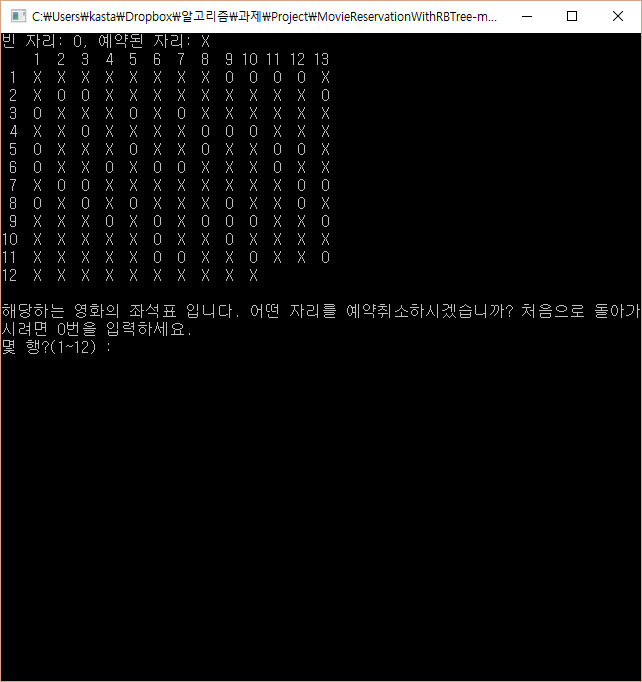
: 영화를 선택하면 그 다음으로 예약을 취소하고 싶은 날을 입력 받는다. 예를 들어 1번 영화를 선택하면 선택한 영화의 이름과 함께 며칠 후에 상영되는 영화의 예약을 취소하고 싶은지 묻는다. 0번을 누르면 처음 프로그램 실행 시 메뉴 출력 화면으로 되돌아간다. 0~6 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 9> 예약 취소 날짜 선택 시 출력 화면

- 예약을 취소할 영화의 시간대 선택

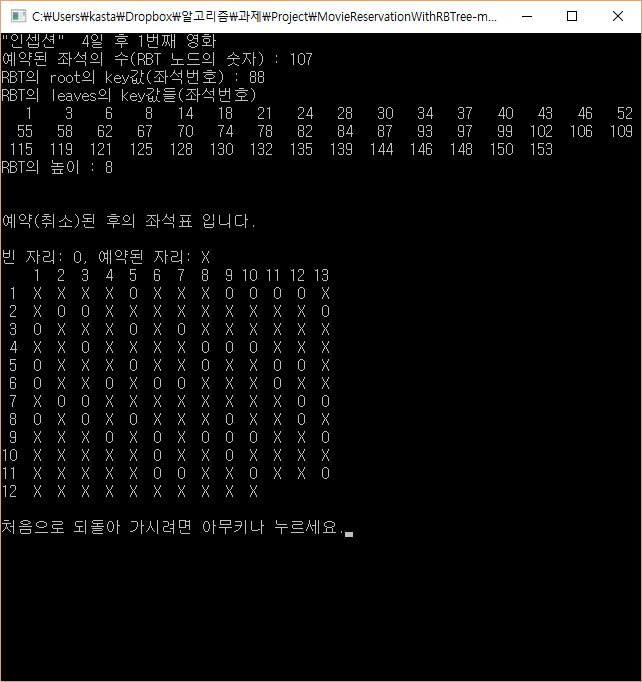
: 예약을 취소하고 싶은 날을 선택 하면 그 날의 영화 상영 시간들이 출력된다. 4일후의 영화를 예약한다고 가정하면 그 영화의 4일후의 상영 시간들이 출력되고 어느 시간대의 예약을 취소하고 싶은지 묻는다. 0번을 누르면 처음 프로그램 실행 시 화면으로 되돌아 간다. 0~3 이외에 다른 번호나 문자를 입력하면 잘못된 입력임을 알리고 다시 번호를 입력 받도록 했다.



<그림 10> 예약 취소 시간 선택 시 출력 화면

- 좌석 선택

: 예약을 취소하고 싶은 시간대를 선택하면 해당 영화의 선택된 상영 시간의 좌석표가 출력된다. 사용자는 이를 보고 예약을 취소하고 싶은 좌석을 선택하고 입력하면 된다. 이때 좌석의 행과 열이 항상 두 자리 수이기 때문에 getch()함수를 사용할 수 가 없어 엔터 키를 입력하면 다음 단계로 진행되도록 했다. 만약 입력 받은 행이나 열이 출력된 좌석표의 행과 열의 범위를 초과하면 잘못된 입력임을 알리고 다시 행과 열을 입력 받도록 했다. 또한 입력 받은 행과 열에 해당하는 좌석이 예약되지 않은 좌석이면 아직 예약되지 않은 좌석임을 알리고 다시 행과 열을 입력 받도록 했다. 올바른 행과 열을 입력(예약된 좌석)하게 되면 예약취소가 성공했음을 알리고 다음단계로 진행된다. 이때 잠시 동안 프로그램의 진행을 지연시켜서 사용자가 예약이 취소되었음을 충분히 인지하도록 했다.



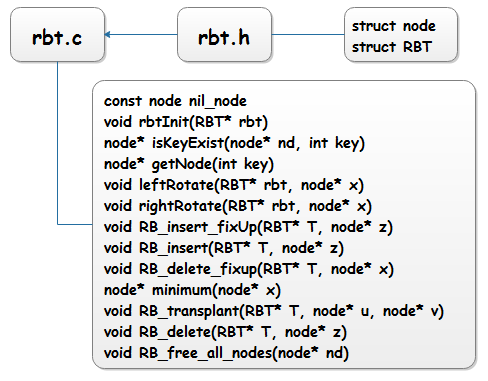
<그림 11> 예약 취소 성공 시 출력 화면

- 예약 취소 성공

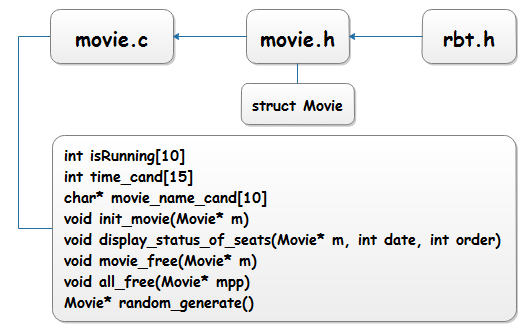
: 예약 취소 과정이 성공적으로 진행되면 <그림 11>과 같은 화면이 출력된다. 어느 영화를 언제, 어느 시간대의 예약이 취소 되었는지 사용자에게 알려주고 그 영화의 RB Tree의 node수와 root의 key값, leaf들, 높이를 순서대로 출력하도록 했다. 마지막으로 예약이 취소된 후의 좌석표를 출력하여 사용자의 예약 취소 과정이 성공적으로 진행 되었음을 알려준다. 그리고 좌석표를 충분히 확인 할 수 있도록 다른 입력을 받기 전까지 다음단계로 진행되지 않도록 했다. 충분히 확인 한 후에 아무 키나 입력하면 처음 프로그램 실행화면으로 되돌아 간다.

1. 모듈 설명

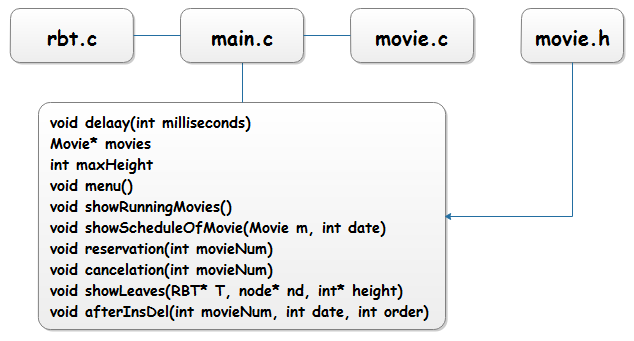
여기서는 소스코드의 모듈에 대한 자세한 설명 대신 전체적인 구조에 대해 설명한다. 자세한 설명은 소스코드 내에 주석으로 처리되어 있다. 화살표는 소스파일이 해당하는 헤더파일을 또는 헤더파일이 다른 헤더파일을 include한다는 것을 나타낸다.



: 영화의 좌석표를 구성하는데 사용되는 알고리즘을 구현하는 코드들이 작성되어 있다. 교과서의 RB Tree 수도코드를 기반으로 작성되었고 처음 Tree를 생성하는 함수, node를 생성하는 함수 그리고 Tree에 해당 key값을 가진 node가 존재하는지 확인하는 함수를 추가했다.

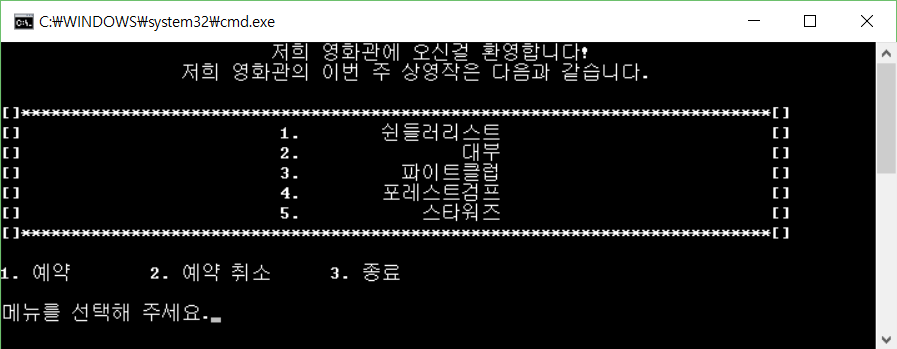


: 영화 구조체와 이것을 기반으로 한 영화관의 정보들을 랜덤으로 생성하는 함수가 작성되어 있다. 모든 과정이 끝나고 메인 함수의 처음에 생성되었던 영화관의 정보들의 메모리를 반환하는 함수 또한 작성되었다.



: 프로그램의 UI, 좌석 예약 그리고 예약 취소를 구현하는 코드들이 작성되어 있다. 영화 구조체 생성을 위해 movie.h를 include 했다.

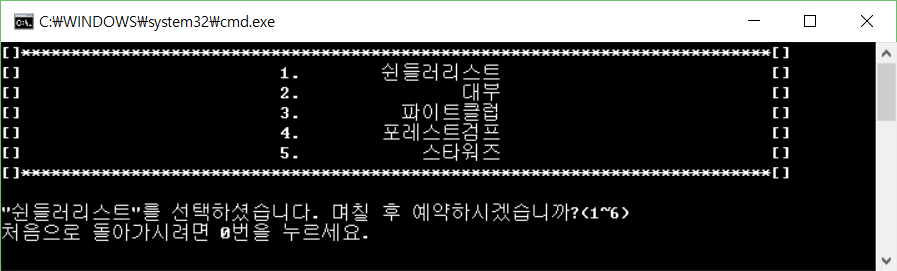
1. Extensive execution examples and analysis
2. 예약 예제



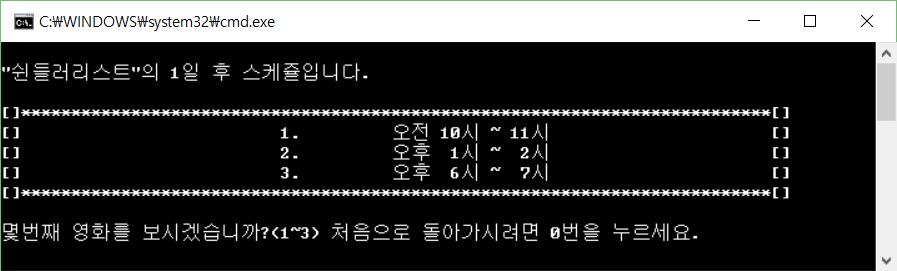
초기화면에서 1번을 누르면 예약화면으로 넘어간다. (엔터키를 따로 누르지 않음) 이 때, 초기화면에서 나오는 영화목록들은 메뉴를 출력하는 menu() 함수 내에서, showRunningMovies() 라는 함수를 호출하여, 이 프로그램을 시작 시에 만들어 놓은 랜덤 영화 정보들 중에서 현재 상영 중인 목록을 표시해 준다.



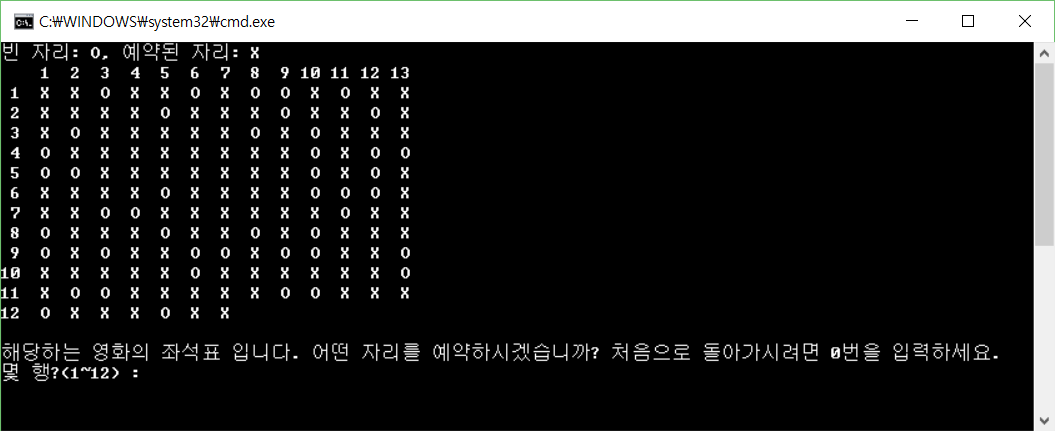
예약화면에서 예약할 영화를 누르게 되는데, 입력받은 숫자에 따라 movie 구조체의 index에 접근하게 된다.



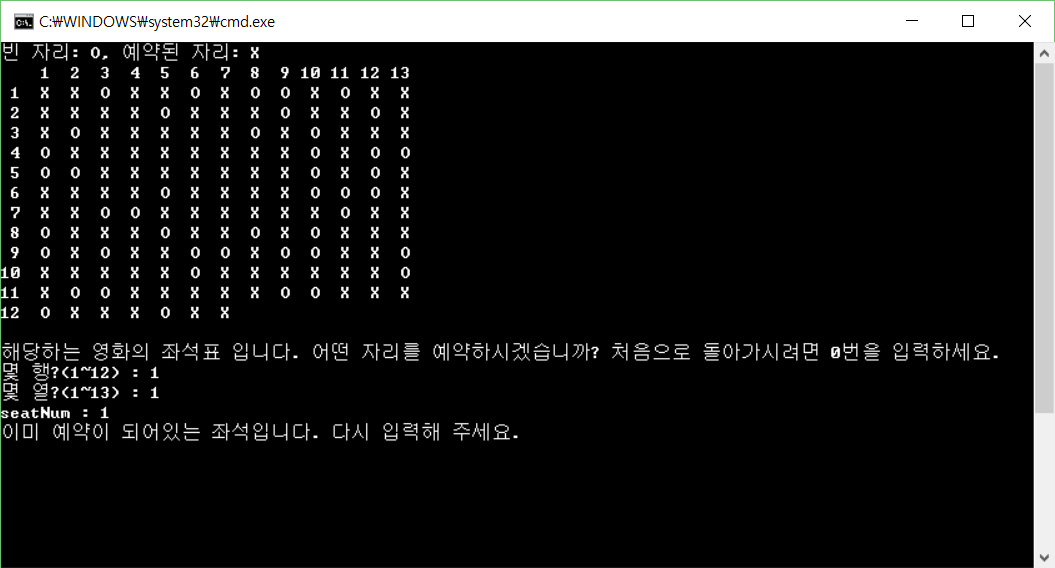
문제에 조건에서 오늘부터 6일까지의 영화를 예약할 수 있도록 만들어야 하는 제약조건에 따라, 선택한 영화에 대하여 1일~6일 사이의 숫자를 입력받게 하고, 그에 따라 해당 영화 구조체의 해당 날짜에 접근한다.



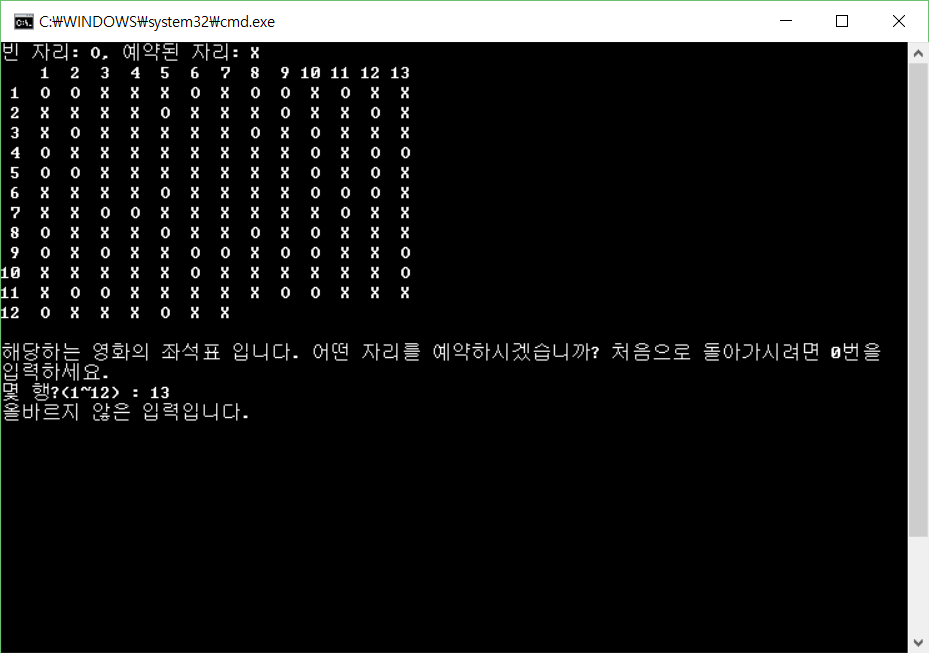
뒤이어 해당 영화, 해당 날짜 중, 상영 시간 목록에서 한 스케쥴을 선택할 수 있도록 한다.



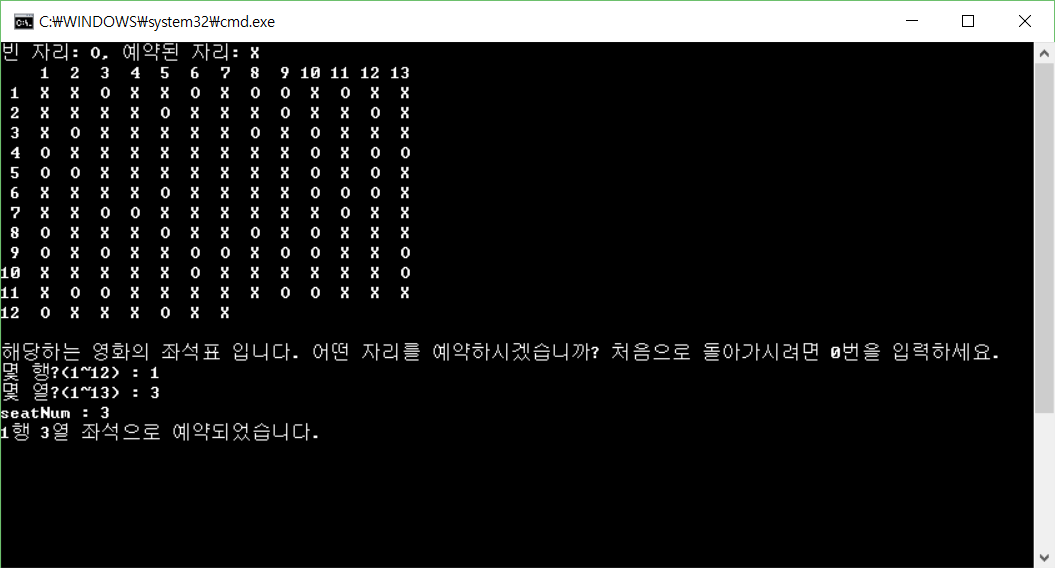
사용자에게서 영화, 날짜, 스케쥴을 입력받은 후에는 그에 해당하는 좌석표를 출력하게 된다. 이는 레드블랙트리를 이용하여 구현되었는데, 레드블랙트리에는 예약된 좌석번호가 key로써 저장되어 있다. 이를 이용하여 예약 좌석 현황을 출력하기 위해, 1부터 총좌석 수까지 레드블랙트리에 해당하는 숫자가 key로써 존재하는지 확인한 후, 존재한다면 예약이 되어있는 좌석으로 인식하고 존재하지 않는다면 빈자리로 인식하여 그에 따라 각각 X / O 로 콘솔창에 나타내 준다.



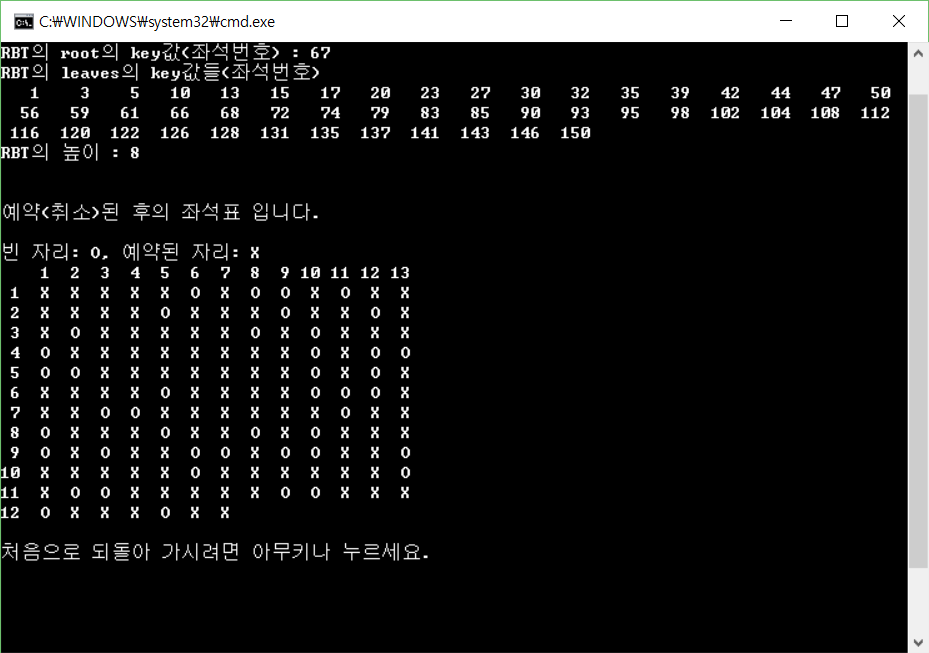
만일 이미 예약되어 있는 좌석을 고르면 위와 같은 메시지를 출력하고 다시 입력을 받는다.



범위를 넘어가는 입력을 할 시에도 올바르지 않은 입력이라는 메시지를 출력 후 다시 입력을 받는다.



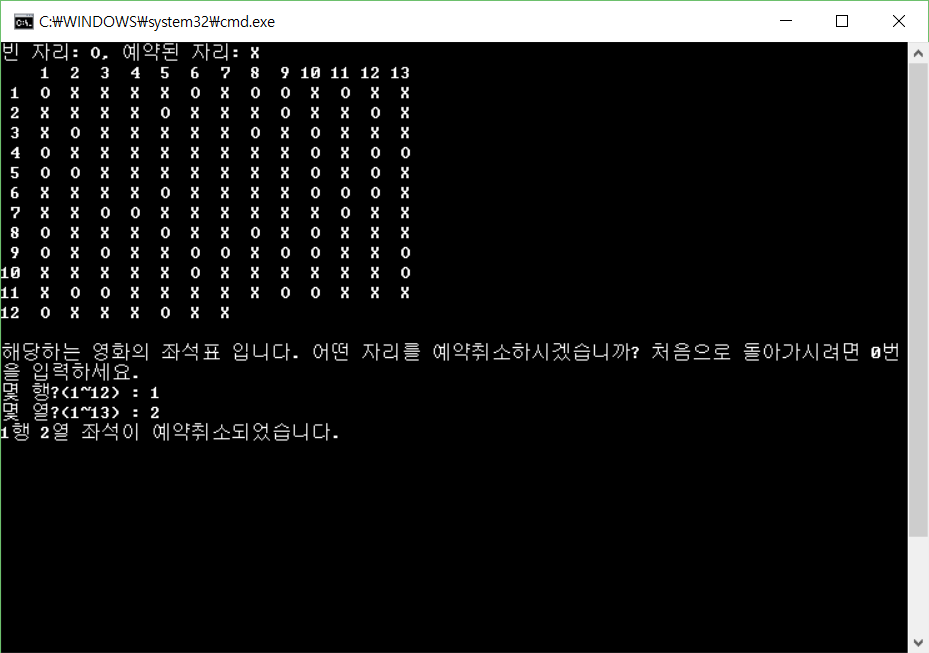
올바른 입력을 받으면 일단 예약이 되었다는 메시지를 표시해 준다. 이 때, 해당 좌석번호가 해당 영화, 날짜, 스케쥴에 해당되는 RBT에 추가되게 된다.



이후, 문제의 추가적인 요구사항에 따라서, 다음과 같은 정보를 출력해 준다.

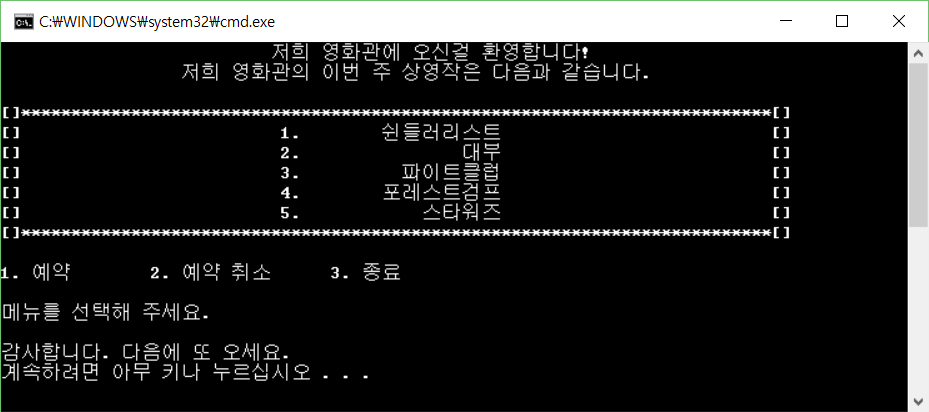
* RBT의 root의 key값
* RBT의 leaves의 key 값들
* RBT의 높이
* 예약된 후의 좌석표

1. 예약 취소 예제



예약 과정과 거의 모든 과정이 비슷하며 다른 부분은 예약 취소가 된 후에, 해당 RBT에서 좌석번호가 없어진다는 점이다.

1. 종료 예제



초기화면에서 3번 키를 누르면, “감사합니다. 다음에 또 오세요.”라는 메시지를 출력 후, 약간의 딜레이 후에 프로그램을 종료하게 된다. 이 후, main()함수에서 all\_free()라는 함수를 호출하여, 각 영화에서 해당 날짜, 스케쥴에 대한 RBT의 모든 노드들을 반환하고, 반환이 완료된 후에는 모든 영화들에 대하여 할당된 메모리를 반환하고 프로그램이 종료되게 된다.

1. Discussion and conclusion

이유비 : 처음에는 Red black tree 가 단순히 BST의 확장형으로 간단하게만 생각했었다. 우리 수업 교재나 강의 자료에 수도코드가 잘 나와있어서 그대로 옮겨놓기만 하면 되고, UI만 빨리 구현하면 금방 끝낼 수 있겠구나 라고 생각했었다. 하지만, 수도코드답게, 그대로 구현하면 오류가 생겼다. 우리 시스템의 첫 버전은 금방 완성되었지만, 그 이후에 UI수정이나 실행과정에서 나오는 여러 오류들을 수정하는데에 더 오랜 시간이 걸렸다. Red black tree 코드가 그리 만만치 않아서 한번 이해하고 죽 수정하고 잠깐 손을 놓고 있다가 다시 오류가 발생하여 코드를 보면 처음부터 다시 해석해야하는 상황이 발생했다. 주석을 적어놓는다고 열심히 적어놓았지만 꽤나 어려웠다. 하지만, 길은규, 최가람 조원의 도움으로 여러 오류들을 찾아내고 수정할 수 있었으며, 길은규 조원은 여러 UI기능들을 추가하고 그 과정 속에서 발견한 오류들도 수정하여 수고를 덜 수 있었다. 짧았지만 고생한 조원들에게 감사할 따름이다.

길은규 : 프로젝트의 주제가 생각보다 간단한 주제여서 쉽게 빨리 끝낼 수 있을 줄 알았다. 하지만 레드블랙 트리를 구현하는데 있어서 교과서의 수도코드를 C언어로 바꾸는 과정이 생각보다 쉽지 않았다. 예상치 못한 오류도 많았고 수도코드에는 없는 추가적인 내용들이 많이 필요해서 시간이 꽤 소요되었다. 하지만 그만큼 레드블랙 트리에 대해 더 자세히 알게 되었고 기말고사를 준비하는 데에도 큰 도움을 준 것 같다. 성공적으로 프로젝트를 끝낼 수 있게 되어 기쁘고 열심히 해 준 조원들에게도 고맙다는 말을 전하고 싶다.

최가람 : 이번에 알고리즘 수업에서 교수님께서부터 Design and implementation of a theater reservation system 구현하는 과제를 부여받고 이 과제를 해결할 수 있을지에 대해서 많은 걱정을 하였습니다. 그렇지만 이번 과제를 통해서 Red Black Tree에 대해서 확실히 이해할 수 있는 기반이 되었습니다. 또 현재 지금 영화관에서 사용하고 있는 예약시스템을 유저 입장으로써 사용만을 해보았다면, 이번 계기로 극장의 영화시스템이  컴퓨터를 이용하여 이렇게 하나하나 코드화가 되어있구나! 이런 생각을 하였습니다. 그리고 이렇게 편리한 시스템을 사용자의 입장에서 사용만 해보았지, 구현해보는 경험을 해보니 쉬운 것이 아님을 다시 한번 느끼는 시간이었습니다. 이번 프로젝트는 학문으로만 배웠던 것이 아닌 실생활에서 사용하고 우리가 흔히 사용하는 것을 해보니, 재미도 있었으며, 흥미도 많이 생기었습니다. 후에 개발자가 된다면 이러한 좋은 시스템을 고객에게 제공하면 많은 편리함을 느낄 수 있게 할 수 있겠다라는 생각도 해보는 좋은 시간이었습니다.

6) Source code

RBT.h

|  |
| --- |
| #ifndef \_RBT\_H\_  #define \_RBT\_H\_  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define COLOR\_RED 0  #define COLOR\_BLACK 1  #define TRUE 1  #define FALSE 0  typedef struct \_node node;  typedef struct \_node {  unsigned key; //key가 0 이면 NULL이라고 가정, RBT는 일반적인 정의이므로, position 보다는 key 라고 이름을 명명  int color;  node \*left, \*right;  node \*parent;  }node;  //RB tree  typedef struct \_rbtree {  node\* root;  node\* nil; //rbt.c 에서 생성될 nil\_node가 들어갈 포인터, 이것으로 red black tree 의 Nil node를 한 개만 생성하여 이용할 수 있다.  int numOfNodes;  } RBT;  void rbtInit(RBT\* rbt);  node\* isKeyExist(node\* nd, int key);  node\* getNode(int key);  void leftRotate(RBT\* rbt, node\* x);  void rightRotate(RBT\* rbt, node\* x);  void RB\_insert(RBT\* T, node\* z);  void RB\_insert\_fixUp(RBT\* T, node\* z);  node\* minimum(node\* x);  void RB\_transplant(RBT\* T, node\* u, node\* v);  void RB\_delete(RBT\* T, node\* z);  void RB\_delete\_fixup(RBT\* T, node\* x);  void RB\_free\_all\_nodes(node\* nd); //이 함수를 사용할 때에는 모든 노드를 순회하며 삭제할 수 있도록 T->root 를 인자로 넣어준다.  #endif |

RBT.c

|  |
| --- |
| #include "rbt.h"  //designated initializer  const node nil\_node = {  .key = 0,  .color = COLOR\_BLACK,  .left = NULL,  .right = NULL,  .parent = NULL  };  //RBT를 사용할 때, root 를 null 로 초기화 하고, 공통으로 사용할 nil을 넣어주는 함수  void rbtInit(RBT\* rbt)  {  rbt->root = &nil\_node; //Todo: 나중에 root 넣을 때, root 의 부모에 &nil\_node 넣어줘야함  rbt->nil = &nil\_node;  rbt->numOfNodes = 0;  }  //찾으려는 key가 존재하는지 확인, 존재한다면 해당노드 반환, 존재하지않는다면 NULL 반환  //key: 찾으려는 key값, curKey: 현재 노드의 key값  node\* isKeyExist(node\* nd, int key) {  if (nd == &nil\_node) return NULL;  int curKey = nd->key;  if (curKey == key) return nd;  else if (curKey > key)  return isKeyExist(nd->left, key);  else  return isKeyExist(nd->right, key);  }  //노드 할당받는 함수  node\* getNode(int key) {  node\* temp = (node\*)malloc(sizeof(node));  temp->key = key;  temp->color = COLOR\_BLACK;  temp->left = &nil\_node;  temp->right = &nil\_node;  temp->parent = &nil\_node;  return temp;  }  //y가 x의 오른쪽 자식일 경우로 시작하여 left rotate 진행  void leftRotate(RBT\* rbt, node\* x) {  node\* y = x->right;  x->right = y->left;  if (y->left != rbt->nil) y->left->parent = x;  y->parent = x->parent;  if (x->parent == rbt->nil) rbt->root = y;  else if (x->parent->left == x) x->parent->left = y;  else x->parent->right = y;  y->left = x;  x->parent = y;  }  //y가 x의 왼쪽 자식일 경우로 시작하여 right rotate 진행  void rightRotate(RBT\* rbt, node\* x) {  node\* y = x->left;  x->left = y->right;  if (y->right != rbt->nil) y->right->parent = x;  y->parent = x->parent;  if (x->parent == rbt->nil) rbt->root = y;  else if (x->parent->right == x) x->parent->right = y;  else x->parent->left = y;  y->right = x;  x->parent = y;  }  //insert 함수에서 z노드는 red로 칠해진 상태로 넘어온다.  //이 때, z->parent->color 가 또 red 라면, RBT의 4번 조건 위반  //위반하는 경우 계속 반복 ==> while(z->parent->color == COLOR\_RED)  void RB\_insert\_fixUp(RBT\* T, node\* z) {  node\* y;  while (z->parent->color == COLOR\_RED)  {  if (z->parent == z->parent->parent->left)  {  y = z->parent->parent->right;  //case 1  if (y->color == COLOR\_RED)  {  z->parent->color = COLOR\_BLACK;  y->color = COLOR\_BLACK;  z->parent->parent->color = COLOR\_RED;  z = z->parent->parent;  }  else  {  //case 2  if (z == z->parent->right)  {  z = z->parent;  leftRotate(T, z);  }  //case 3  z->parent->color = COLOR\_BLACK;  z->parent->parent->color = COLOR\_RED;  rightRotate(T, z->parent->parent);  }  }  //same as then clause with "right" and "left" exchanged.  else  {  y = z->parent->parent->left;  //case 1  if (y->color == COLOR\_RED)  {  z->parent->color = COLOR\_BLACK;  y->color = COLOR\_BLACK;  z->parent->parent->color = COLOR\_RED;  z = z->parent->parent;  }  else  {  //case 2  if (z == z->parent->left)  {  z = z->parent;  rightRotate(T, z);  }  //case 3  z->parent->color = COLOR\_BLACK;  z->parent->parent->color = COLOR\_RED;  leftRotate(T, z->parent->parent);  }  }  }  T->root->color = COLOR\_BLACK;  }  void RB\_insert(RBT\* T, node\* z) {  T->numOfNodes++;  node \*y = T->nil;  node \*x = T->root;  while (x != T->nil)  {  y = x;  //BST와 비슷, 현재 위치의 key보다 찾으려는 key가 작다면 왼쪽자식으로, 크다면 오른쪽 자식으로 이동  if (z->key < x->key)  x = x->left;  else  x = x->right;  }  z->parent = y;  if (y == T->nil)  T->root = z;  else if (z->key < y->key)  y->left = z;  else  y->right = z;  z->left = T->nil;  z->right = T->nil;  z->color = COLOR\_RED; //삽입하는 노드의 색깔을 RED로 칠한 뒤에 insertFixUp( ) 함수를 호출한다.  RB\_insert\_fixUp(T, z);  }  void RB\_delete\_fixup(RBT\* T, node\* x)  {  node \*w,\*temp = NULL;  while (x != T->root && x->color == COLOR\_BLACK)  {  if (x == x->parent->left)  {  if(x->parent->right != T->nil) w = x->parent->right;  else {  temp = getNode(0);  temp->parent = x->parent;  x->parent->right = temp;  temp->left = T->nil;  temp->right = T->nil;  w = temp;  }  //case 1  if (w->color == COLOR\_RED)  {  w->color = COLOR\_BLACK;  x->parent->color = COLOR\_RED;  leftRotate(T, x->parent);  w = x->parent->right;  }  //case 2  if (w->left->color == COLOR\_BLACK && w->right->color == COLOR\_BLACK)  {  w->color = COLOR\_RED;  x = x->parent;  }  else  {  //case 3  if (w->right->color == COLOR\_BLACK)  {  w->left->color = COLOR\_BLACK;  w->color = COLOR\_RED;  rightRotate(T, w);  w = x->parent->right;  }  //case 4  w->color = x->parent->color;  x->parent->color = COLOR\_BLACK;  w->right->color = COLOR\_BLACK;  leftRotate(T, x->parent);  x = T->root;  }  }  //same as then clause with "right" and "left" exchanged.  else  {  if (x->parent->left != T->nil) w = x->parent->left;  else {  temp = getNode(0);  temp->parent = x->parent;  x->parent->right = temp;  temp->left = T->nil;  temp->right = T->nil;  w = temp;  }  //case 1  if (w->color == COLOR\_RED)  {  w->color = COLOR\_BLACK;  x->parent->color = COLOR\_RED;  rightRotate(T, x->parent);  w = x->parent->left;  }  //case 2  if (w->right->color == COLOR\_BLACK && w->left->color == COLOR\_BLACK)  {  w->color = COLOR\_RED;  x = x->parent;  }  else  {  //case 3  if (w->left->color == COLOR\_BLACK)  {  w->right->color = COLOR\_BLACK;  w->color = COLOR\_RED;  leftRotate(T, w);  w = x->parent->left;  }  //case 4  w->color = x->parent->color;  x->parent->color = COLOR\_BLACK;  w->left->color = COLOR\_BLACK;  rightRotate(T, x->parent);  x = T->root;  }  }  }  if (temp != NULL) {  if (temp->parent->left == temp) temp->parent->left = T->nil;  else temp->parent->right = T->nil;  free(temp);  }  x->color = COLOR\_BLACK;  }  node\* minimum(node\* x) {  if (x->left->key == 0)  return x;  else return minimum(x->left);  }  //RB\_transplant ( ) 함수는 v.left 나 v.right 를 update 하지 않는다.  //그 부분은 이 함수의 호출자가 책임진다.  //현재 u와 u의 부모와의 관계를 v와의 관계로 바꾼다.  //v의 부모가 u의 부모가 되고, u의 부모의 자식은 더이상 u가 아니라 v가 된다.  //v는 nil\_node 일 수 있다.  void RB\_transplant(RBT\* T, node\* u, node\* v)  {  if (u->parent == T->nil)  T->root = v;  else if (u == u->parent->left)  u->parent->left = v;  else  u->parent->right = v;  if (v != T->nil)  v->parent = u->parent;  }  void RB\_delete(RBT\* T, node\* z)  {  T->numOfNodes--;  node\* x;  node\* y = z;  node\* temp = NULL;  int y\_original\_color = y->color;  if (z->left == T->nil)  {  x = z->right;  RB\_transplant(T, z, z->right);  if (z->right == T->nil) {  temp = getNode(0);  if (z->parent->right == z) z->parent->right = temp;  else z->parent->left = temp;  temp->parent = z->parent;  x = temp;  }  }  else if (z->right == T->nil)  {  x = z->left;  RB\_transplant(T, z, z->left);  }  else  {  //이 부분에서 z는 삭제할 노드를, y는 삭제되고 그 자리를 채울 노드를 나타낸다.  //z->left 와 z->right 가 T->nil 이 아니므로, 자식은 반드시 존재한다. 따라서, minimum 함수를 사용할 수 있다.  //successor 를 사용할 필요가 없다. 자식이 있으므로, 부모를 찾아갈 필요가 없다.  y = minimum(z->right);  y\_original\_color = y->color;  if (y->right == T->nil) {  temp = getNode(0);  y->right = temp;  temp->parent = y;  x = temp;  }  else x = y->right;  //없애려는 노드(==z)의 successor(==y) 가 바로 오른쪽 자식인 경우  //(x는 nil\_node 이면 안된다.)  if (y->parent == z) x->parent = y;  //그렇지 않은 경우  else  {  RB\_transplant(T, y, y->right);  y->right = z->right; //z자리로 y를 채우기 위해, y의 오른쪽 자식을 z의 오른쪽 자식을 가리키게 한다.  if(y->right != T->nil) y->right->parent = y; //바뀐 y의 오른쪽 자식의 부모를 y를 가리키게 한다.(아직 z를 가리키고 있으므로)  }  //지금까지 과정으로 삭제할 노드의 오른쪽 자식트리를 이용하여 z자리를 채워 줬고, 새로 채운 노드와 삭제되는 노드의 왼쪽 자식트리를 이어주는 과정을 진행한다.  RB\_transplant(T, z, y); //z 자리에 y를 채워주기 위해서 z의 부모와 z의 관계를 y와의 관계로 바꿔준다.  y->left = z->left; //y의 왼쪽 자식을 z의 왼쪽 자식으로 채운다.  y->left->parent = y; //y->left 가 nil\_node 일 수는 없다. 앞의 if 조건문에서 처리가 되었기 때문이다. 따라서, y->left->parent 접근은 valid하다.  y->color = z->color; //색깔을 일치시킨다.  }  if (y\_original\_color == COLOR\_BLACK) //if 조건문에 따라 y가 z노드를 가리킬 수도 있고, 새로 채워질 노드를 가리킬 수도 있다. 경우에 따른 y에 대하여 그 색깔을 나타낸다.  RB\_delete\_fixup(T, x);  if (temp != NULL) {  if (temp->parent->left == temp) temp->parent->left = T->nil;  else temp->parent->right = T->nil;  free(temp);  }  free(z); //제거된 노드의 메모리 해제  }  //할당해준 모든 노드를 반환하기 위한 함수  void RB\_free\_all\_nodes(node\* nd)  {  if (nd == &nil\_node) return;  //post-order traversal  RB\_free\_all\_nodes(nd->left);  RB\_free\_all\_nodes(nd->right);  free(nd);  } |

movie.h

|  |
| --- |
| #ifndef \_MOVIE\_H\_  #define \_MOVIE\_H\_  #include "rbt.h"  #include <string.h>  #define TOTAL\_NUM\_OF\_MOVIE 5 //총 영화의 수  //영화관 구조체  typedef struct \_movie  {  char\* name;  RBT seats[6][3]; //각각의 영화에 대한 좌석 정보를 담고 있다. 이 안에 들어가 있는 노드들의 key는 예약된 좌석번호를 나타낸다.  int sched[6][3]; //[x][y] 라 하였을 때, x일 뒤의 이 영화 중 y번째 영화의 시작시간을 담고 있다. (정시에 시작하는 것으로 가정)  int num\_of\_seats[6][3]; //영화관마다 자리개수  } Movie;  //movie 구조체 초기화 함수  void init\_movie(Movie\* m);  /\*  \* 영화좌석 display 함수  \* 해당 영화의 구조체 주소, 날짜, 그 날의 그 영화에 관해서 몇번째 영화인지를 나타내는 변수 세 개를 넘긴다.  \* 그리고 그 영화의 좌석 현황을 시각적으로 보여준다.  \*/  void display\_status\_of\_seats(Movie\* m, int date, int order);  //movie 구조체 내의 RB\_tree 에 대한 free 함수  void movie\_free(Movie\* m);  //랜덤하게 생성된 총 영화의 해제를 담당하는 함수, 내부에서 movie\_free() 함수를 호출한다.  void all\_free(Movie\* mpp);  /\*  \* 랜덤좌석 생성 부분  \* 랜덤 요소:  \* 영화이름, 상영시간, 주간 스케쥴(6일치의 영화이름, 상영시간을 생성하면 충족)  \* 영화의 총 자리 개수(100~300사이의 값)  \* 이미 차있는 자리(RB\_tree 에 삽입, 70% 이상)  \*/  Movie\* random\_generate();  #endif |

movie.c

|  |
| --- |
| #include "movie.h"  //랜덤 생성할 영화 이름  char\* movie\_name\_cand[10] = {  "쇼생크탈출", "대부", "다크나이트", "쉰들러리스트", "펄프픽션",  "반지의제왕", "파이트클럽", "스타워즈", "포레스트검프", "인셉션"  };  int isRunning[10]; //random\_generate() 함수에서 똑같은 이름의 영화를 중복해서 생성하지 않기 위해 사용되는 변수  //13을 넘어가면 12를 빼고 '오후'를 붙여줘서 사용  //시간 중복처리를 위한 flag 변수는 랜덤생성하는 함수 내에서 적용  int time\_cand[15] = {  8, 9, 10, 11, 12,  13, 14, 15, 16, 17,  18, 19, 20, 21, 22  };  //Movie 구조체를 사용하기 전에 초기화를 진행  void init\_movie(Movie\* m)  {  m->name = NULL;  for (int i = 0; i < 6; i++)  for (int j = 0; j < 3; j++)  {  rbtInit(&m->seats[i][j]);  m->sched[i][j] = 0;  m->num\_of\_seats[i][j] = 0;  }  }  void display\_status\_of\_seats(Movie\* m, int date, int order)  {  printf("빈 자리: O, 예약된 자리: X\n");  int seats = m->num\_of\_seats[date][order];  printf(" ");  for (int i = 0; i\*i < seats; i++)  printf("%2d ", i + 1);  printf("\n%2d ", 1);  //RB\_tree 를 이용하여 num\_of\_seats 만큼 반복하여 해당 자리가 RB\_tree안에 있으면 X 없으면 O 출력  for (int i = 1, j = 1, count = 1; i <= seats; i++, j++)  {  //TODO: 이 자리에서 RB\_tree 체크 후, O, X 중에 하나를 print  printf("%2c ", isKeyExist(m->seats[date][order].root, i) ? 'X' : 'O');  //적당히 가로세로 길이를 나누기 위한 부분  //자리가 100~300자리로 변할 수 있으므로 절대적인 숫자로 한 줄씩 라인피드하면 안된다.  //i !=seats 조건은 만일 해당 영화의 좌석의 총 수가 정확히 어떤 수의 제곱이라면(n^2), n+1행이 내용없이 추가되는 버그가 생긴다.  //따라서, i==seats 인 경우에는 printf("\n%2 ", count); 가 실행되지 않도록 한다.  if (j\*j >= seats && i !=seats)  {  count++;  printf("\n%2d ", count); //세로로 문자출력하기 위한 부분.  j = 0;  }  }  }  void movie\_free(Movie\* m)  {  for (int i = 0; i < 6; i++)  for (int j = 0; j < 3; j++)  RB\_free\_all\_nodes(m->seats[i][j].root);  }  void all\_free(Movie\* mpp)  {  for (int i = 0; i < TOTAL\_NUM\_OF\_MOVIE; i++)  movie\_free(&mpp[i]);  free(mpp);  }  Movie\* random\_generate()  {  //영화를 선택한 후에, isRunning[] 변수도 업데이트하여 이미 선택된 영화를 중복선택하지 않도록 한다.  srand((unsigned)time(NULL));  Movie\* gen = (Movie\*)malloc(sizeof(Movie) \* 5);  for (int i = 0; i < 5; i++)  init\_movie(&gen[i]); //Movie 구조체 사용 전 초기화  //영화이름 정하기  for (int i = 0; i < 5; i++)  {  int idx;  while (1)  {  idx = rand() % 10;  if (isRunning[idx] == FALSE)  {  isRunning[idx] = TRUE;  gen[i].name = movie\_name\_cand[idx];  break;  }  }  }  //영화이름이 중복되지 않도록 사용하기 위한 변수  //하루 사이에서만 중복되지 않게 적용  //주간 스케쥴 내에서는 중복되어도 됨(실제 영화관에서도 같은 영화가 주간에 여러 번 상영하기도 하므로)  int is\_movie\_selected[10] = { 0, };  //시간대가 중복되지 않도록 사용(사실, 영화 상영실이 여러 개라면, 시간이 중복되어도 상관은 없지만 그냥 중복되지 않게 한다고 가정)  int is\_time\_selected[15] = { 0, };  //스케쥴 정하기  //i: 6일치 j: 5개의 영화 k: 하루의 3개 스케쥴 --> 5개의 영화가 하루에 스케쥴이 모두 안겹치게 함.  for (int i = 0; i < 6; i++)  {  memset(is\_time\_selected, 0, 15 \* sizeof(int)); //6일 중, 시간대가 하루 중에서만 안겹치면 되므로, 6일치를 만드는 해당 loop 의 처음 부분에 초기화를 진행한다.  for (int j = 0; j < 5; j++) {  for (int k = 0; k < 3; k++)  {  int idx;  while (1)  {  idx = rand() % 15;  if (is\_time\_selected[idx] == FALSE)  {  is\_time\_selected[idx] = TRUE;  gen[j].sched[i][k] = time\_cand[idx];  break;  }  }  }  //스케쥴 중 빠른 시간이 앞으로 오도록 변경  int swap;  if (gen[j].sched[i][0] > gen[j].sched[i][1]) {  swap = gen[j].sched[i][0];  gen[j].sched[i][0] = gen[j].sched[i][1];  gen[j].sched[i][1] = swap;  }  if (gen[j].sched[i][1] > gen[j].sched[i][2]) {  swap = gen[j].sched[i][1];  gen[j].sched[i][1] = gen[j].sched[i][2];  gen[j].sched[i][2] = swap;  }  if (gen[j].sched[i][0] > gen[j].sched[i][1]) {  swap = gen[j].sched[i][0];  gen[j].sched[i][0] = gen[j].sched[i][1];  gen[j].sched[i][1] = swap;  }  }  }  //각 영화의 좌석 수 생성하기  for (int i = 0; i < 5; i++)  for (int j = 0; j < 6; j++)  for (int k = 0; k < 3; k++)  gen[i].num\_of\_seats[j][k] = rand() % 201 + 100; //100~300 사이의 범위  //각 영화의 좌석 수의 70%만큼 RBT에 삽입하기  int isDuplicate[301];  for (int i = 0; i < 5; i++)  {  for (int j = 0; j < 6; j++)  {  for (int k = 0; k < 3; k++)  {  memset(isDuplicate, 0, 301 \* sizeof(int));  int numOfSeats = gen[i].num\_of\_seats[j][k];  for (int l = 0; l < numOfSeats \* 7 / 10; l++) //70% 생성  {  int idx;  while (1)  {  idx = rand() % numOfSeats + 1; //numOfSeats 가 100자리라면, 1~100번째 까지의 숫자가 랜덤으로 생성됨  if (isDuplicate[idx] == FALSE) //만일 중복된 숫자가 안나온다면, isDuplicate[] 을 update한 후에, loop를 빠져나가 해당 자리에 대한 노드를 red black tree에 삽입해 준다.  {  isDuplicate[idx] = TRUE;  break;  }  }  node\* ins = getNode(idx);  RB\_insert(&gen[i].seats[j][k], ins);  }  }  }  }  //5개의 Movie 구조체를 할당하고, 첫번째 Movie를 가리키는 pointer를 반환한다.  return gen;  } |

main.c

|  |
| --- |
| #include "movie.h"  //랜덤 생성할 영화 이름  char\* movie\_name\_cand[10] = {  "쇼생크탈출", "대부", "다크나이트", "쉰들러리스트", "펄프픽션",  "반지의제왕", "파이트클럽", "스타워즈", "포레스트검프", "인셉션"  };  int isRunning[10]; //random\_generate() 함수에서 똑같은 이름의 영화를 중복해서 생성하지 않기 위해 사용되는 변수  //13을 넘어가면 12를 빼고 '오후'를 붙여줘서 사용  //시간 중복처리를 위한 flag 변수는 랜덤생성하는 함수 내에서 적용  int time\_cand[15] = {  8, 9, 10, 11, 12,  13, 14, 15, 16, 17,  18, 19, 20, 21, 22  };  //Movie 구조체를 사용하기 전에 초기화를 진행  void init\_movie(Movie\* m)  {  m->name = NULL;  for (int i = 0; i < 6; i++)  for (int j = 0; j < 3; j++)  {  rbtInit(&m->seats[i][j]);  m->sched[i][j] = 0;  m->num\_of\_seats[i][j] = 0;  }  }  void display\_status\_of\_seats(Movie\* m, int date, int order)  {  printf("빈 자리: O, 예약된 자리: X\n");  int seats = m->num\_of\_seats[date][order];  printf(" ");  for (int i = 0; i\*i < seats; i++)  printf("%2d ", i + 1);  printf("\n%2d ", 1);  //RB\_tree 를 이용하여 num\_of\_seats 만큼 반복하여 해당 자리가 RB\_tree안에 있으면 X 없으면 O 출력  for (int i = 1, j = 1, count = 1; i <= seats; i++, j++)  {  //TODO: 이 자리에서 RB\_tree 체크 후, O, X 중에 하나를 print  printf("%2c ", isKeyExist(m->seats[date][order].root, i) ? 'X' : 'O');  //적당히 가로세로 길이를 나누기 위한 부분  //자리가 100~300자리로 변할 수 있으므로 절대적인 숫자로 한 줄씩 라인피드하면 안된다.  //i !=seats 조건은 만일 해당 영화의 좌석의 총 수가 정확히 어떤 수의 제곱이라면(n^2), n+1행이 내용없이 추가되는 버그가 생긴다.  //따라서, i==seats 인 경우에는 printf("\n%2 ", count); 가 실행되지 않도록 한다.  if (j\*j >= seats && i !=seats)  {  count++;  printf("\n%2d ", count); //세로로 문자출력하기 위한 부분.  j = 0;  }  }  }  void movie\_free(Movie\* m)  {  for (int i = 0; i < 6; i++)  for (int j = 0; j < 3; j++)  RB\_free\_all\_nodes(m->seats[i][j].root);  }  void all\_free(Movie\* mpp)  {  for (int i = 0; i < TOTAL\_NUM\_OF\_MOVIE; i++)  movie\_free(&mpp[i]);  free(mpp);  }  //TODO: 나중에 main 에서 이 함수 반환된 포인터 반환필요  Movie\* random\_generate()  {  //영화를 선택한 후에, isRunning[] 변수도 업데이트하여 이미 선택된 영화를 중복선택하지 않도록 한다.  srand((unsigned)time(NULL));  Movie\* gen = (Movie\*)malloc(sizeof(Movie) \* 5);  for (int i = 0; i < 5; i++)  init\_movie(&gen[i]); //Movie 구조체 사용 전 초기화  //영화이름 정하기  for (int i = 0; i < 5; i++)  {  int idx;  while (1)  {  idx = rand() % 10;  if (isRunning[idx] == FALSE)  {  isRunning[idx] = TRUE;  gen[i].name = movie\_name\_cand[idx];  break;  }  }  }  //영화이름이 중복되지 않도록 사용하기 위한 변수  //하루 사이에서만 중복되지 않게 적용  //주간 스케쥴 내에서는 중복되어도 됨(실제 영화관에서도 같은 영화가 주간에 여러 번 상영하기도 하므로)  int is\_movie\_selected[10] = { 0, };  //시간대가 중복되지 않도록 사용(사실, 영화 상영실이 여러 개라면, 시간이 중복되어도 상관은 없지만 그냥 중복되지 않게 한다고 가정)  int is\_time\_selected[15] = { 0, };  //스케쥴 정하기  //i: 6일치 j: 5개의 영화 k: 하루의 3개 스케쥴 --> 5개의 영화가 하루에 스케쥴이 모두 안겹치게 함.  for (int i = 0; i < 6; i++)  {  memset(is\_time\_selected, 0, 15 \* sizeof(int)); //6일 중, 시간대가 하루 중에서만 안겹치면 되므로, 6일치를 만드는 해당 loop 의 처음 부분에 초기화를 진행한다.  for (int j = 0; j < 5; j++) {  for (int k = 0; k < 3; k++)  {  int idx;  while (1)  {  idx = rand() % 15;  if (is\_time\_selected[idx] == FALSE)  {  is\_time\_selected[idx] = TRUE;  gen[j].sched[i][k] = time\_cand[idx];  break;  }  }  }  //스케쥴 중 빠른 시간이 앞으로 오도록 변경  int swap;  if (gen[j].sched[i][0] > gen[j].sched[i][1]) {  swap = gen[j].sched[i][0];  gen[j].sched[i][0] = gen[j].sched[i][1];  gen[j].sched[i][1] = swap;  }  if (gen[j].sched[i][1] > gen[j].sched[i][2]) {  swap = gen[j].sched[i][1];  gen[j].sched[i][1] = gen[j].sched[i][2];  gen[j].sched[i][2] = swap;  }  if (gen[j].sched[i][0] > gen[j].sched[i][1]) {  swap = gen[j].sched[i][0];  gen[j].sched[i][0] = gen[j].sched[i][1];  gen[j].sched[i][1] = swap;  }  }  }  //각 영화의 좌석 수 생성하기  for (int i = 0; i < 5; i++)  for (int j = 0; j < 6; j++)  for (int k = 0; k < 3; k++)  gen[i].num\_of\_seats[j][k] = rand() % 201 + 100; //100~300 사이의 범위  //각 영화의 좌석 수의 70%만큼 RBT에 삽입하기  int isDuplicate[301];  for (int i = 0; i < 5; i++)  {  for (int j = 0; j < 6; j++)  {  for (int k = 0; k < 3; k++)  {  memset(isDuplicate, 0, 301 \* sizeof(int));  int numOfSeats = gen[i].num\_of\_seats[j][k];  for (int l = 0; l < numOfSeats \* 7 / 10; l++) //70% 생성  {  int idx;  while (1)  {  idx = rand() % numOfSeats + 1; //numOfSeats 가 100자리라면, 1~100번째 까지의 숫자가 랜덤으로 생성됨  if (isDuplicate[idx] == FALSE) //만일 중복된 숫자가 안나온다면, isDuplicate[] 을 update한 후에, loop를 빠져나가 해당 자리에 대한 노드를 red black tree에 삽입해 준다.  {  isDuplicate[idx] = TRUE;  break;  }  }  node\* ins = getNode(idx);  RB\_insert(&gen[i].seats[j][k], ins);  }  }  }  }  //5개의 Movie 구조체를 할당하고, 첫번째 Movie를 가리키는 pointer를 반환한다.  return gen;  } |