Report3

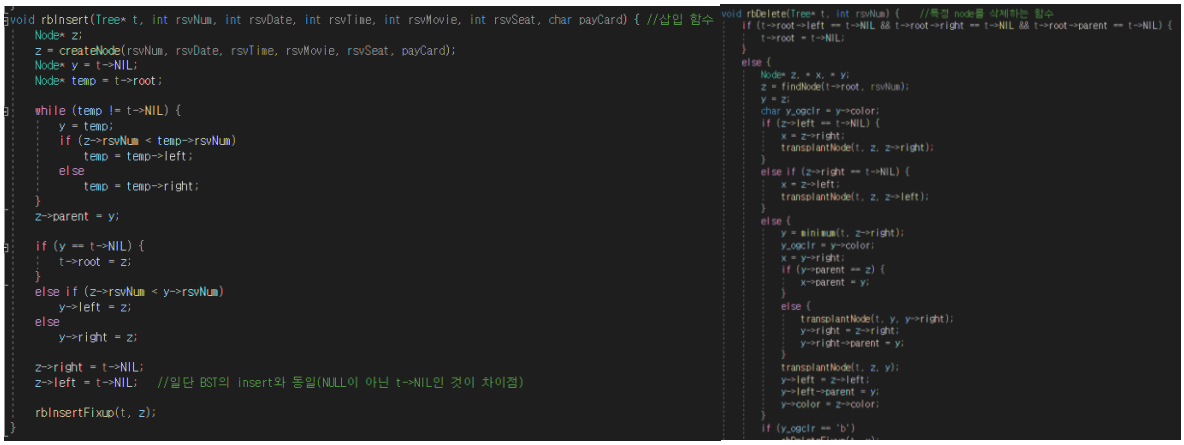
Multiplex theater reservation system

2016314364 박수헌

1. Introduction

이 프로그램은 간단한 영화관 예매 시스템입니다. 언어는 c언어로 구성되어 있으며, 크게 총 세 단계의 레벨로 구성돼 있습니다. 우선 가장 위의 레벨에는 UI가 있습니다. 글씨들로 구성했으며, 밑에 유저가 원하는 기능의 번호를 입력 해 실행하게 되는 방식입니다. 두 번째 레벨은 예매, 예매 취소, 총 노드 수 등등의 예매 시스템을 구현하기 위한 함수들로 구성돼 있고, 마지막 레벨은 자료구조입니다. 자료구조는 rb-tree로 구성했으며, 총 세 개의 rb-tree를 만들었습니다.

Rb-tree의 insert와 delete함수의 인자들은 아래와 같습니다.



```
void rbInsert(Tree* t, int rsvNum, int rsvDate, int rsvTime, int rsvMovie, int rsvSeat, char payCard) { //삽입 함수
    Node* z;
    z = createNode(rsvNum, rsvDate, rsvTime, rsvMovie, rsvSeat, payCard);
    Node* y = t->NIL;
    Node* temp = t->root;

    while (temp != t->NIL) {
        y = temp;
        if (z->rsvNum < temp->rsvNum)
            temp = temp->left;
        else
            temp = temp->right;
    }
    z->parent = y;

    if (y == t->NIL) {
        t->root = z;
    }
    else if (z->rsvNum < y->rsvNum)
        y->left = z;
    else
        y->right = z;

    z->right = t->NIL;
    z->left = t->NIL;   //일단 BST의 insert와 동일(NULL이 아닌 t->NIL인 것이 차이점)

    rbInsertFixup(t, z);
}
```

```
void rbDelete(Tree* t, int rsvNum) {    //특정 node를 삭제하는 함수
    if (t->root->left == t->NIL && t->root->right == t->NIL && t->root->parent == t->NIL) {
        t->root = t->NIL;
    }
    else {
        Node* z, * x, * y;
        z = findNode(t->root, rsvNum);
        y = z;
        char y_ogclr = y->color;
        if (z->left == t->NIL) {
            x = z->right;
            transplantNode(t, z, z->right);
        }
        else if (z->right == t->NIL) {
            x = z->left;
            transplantNode(t, z, z->left);
        }
        else {
            y = minimum(t, z->right);
            y_ogclr = y->color;
            x = y->right;
            if (y->parent == z) {
                x->parent = y;
            }
            else {
                transplantNode(t, y, y->right);
                y->right = z->right;
                y->right->parent = y;
            }
            transplantNode(t, z, y);
            y->left = z->left;
            y->left->parent = y;
            y->color = z->color;
        }
        if (y_ogclr == 'b')
```

rbInsert함수는 예매할 때 사용하는 함수입니다. 따라서 인자로는 모든 정보가 들어가야 합니다. 예약번호, 예약일자, 예약 시간, 예약 영화, 예약 좌석, 결제 방법이 있습니다.

Rbdelete는 예매 취소할 때 사용하는 함수입니다. 예매를 취소할 때는 예약 번호만 입력하면 되기 때문에 인자로는 예약 번호밖에 필요하지 않습니다.

Rb-tree의 정렬을 위해 사용되는 값은 예약 번호입니다.

총 3일(01.12.2020, 02.12.2020, 03.12.2020), 3가지의 영화(About time, The Avengers, Frozen), 3가지의 시간(12:00, 15:00, 18:00)에 예매가 가능합니다. 그리고 rb-tree는 각 일자 당 하나씩 만들었습니다. 각 일마다 day1_t, day2_t, day3_t의 rb-tree 자료구조를 가지고 있습니다. 이 tree의 node들이 가지고 있는 정보는 아래와 같습니다.

```
typedef struct Node {
    int rsvNum;
    int rsvDate;
    int rsvTime;
    int rsvMovie;
    int rsvSeat;
    char payCard;
    char color;
    struct Node* right;
    struct Node* left;
    struct Node* parent;
}Node;
```

rsvNum은 예약번호,

rsvDate는 예약한 영화의 일자,

rsvTime은 예약한 영화의 시간,

rsvMovie는 예약한 영화,

rsvSeat은 예약한 좌석,

payCard는 결제를 카드로 하면 'y', 아니라면(현금으로 한다면), 'n'을 담고,

color는 red나 black 중 하나의 색을 가지게 됩니다.

예매를 할 때, 순서대로 날짜, 영화, 시간, 좌석, 결제수단을 선택하게 됩니다. 선택이 완료되면, 이 정보들을 가진 node를 알맞은 날짜의 rb-tree에 rbInsert 함수를 사용해 삽입하게 됩니다.
삽입하면서, 예약번호가 같이 저장됩니다. 이 예약번호는 아래와 같이 생성됩니다.

```
int number[2700];
number[0] = rand() % 20000 + 10000;
for (int i = 0; i < 2700; i++) {
    number[i] = rand() % 20000 + 10000;
    for (int j = 0; j < i; j++) {
        if (number[i] == number[j])
            i--;
    }
}
```

number라는 정수 배열에 10000 ~ 30000 사이의 난수를 생성해 담게 됩니다. 총 3일, 3편, 3타임에 각각 100좌석씩을 가지고 있으니, 모든 예약이 차게 됐을 때 2700개의 예약번호가 필요하므로, 배열의 크기는 2700으로 설정했습니다. 예약번호가 중복 돼서는 안되므로 for문은 중복된 예약번호가 생성된다면 이를 삭제하고 다시 생성해내는 코드입니다. 따라서, 예약을 할 때 이 배열의 첫 index부터 하나씩 가져다 예약번호로 배정하는 구조입니다.

예약을 할 때, 남아있는 유저에게 좌석 정보를 보여주어야 합니다. 이 좌석 정보는

int* seatLayout 배열에 담았습니다. 초기에는 1~100의 숫자를 가지고 있습니다. 그리고 유저가 일자, 영화, 시간을 모두 선택 완료할 경우 아래의 함수를 이용해 남은 좌석을 볼 수 있게 합니다.

```
void inorderSeat(Tree* t, Node* n, int movie, int times, int* seatLayout) {
    if (n != t->NIL) {
        if (n->rsvMovie == movie && n->rsvTime == times) {
            seatLayout[n->rsvSeat - 1] = 0;
        }
        inorderSeat(t, n->left, movie, times, seatLayout);
        inorderSeat(t, n->right, movie, times, seatLayout);
    }
}
```

우선 위의 inorderSeat 함수를 이용해, 해당 날짜의 rb-tree를 중위순회(inorder)하며, 유저가 선택한 영화와 시간이 노드의 rsvMovie, rsvTime 값이 과 일치할 때, 해당 노드의 rsvSeat 값을

seatLayout 배열에서 0으로 바꿔줍니다.

```
void printSeatLayout(int* seatLayout) {
    printf("₩n₩n                                        [ S C R E E N ]₩n₩n₩n");
    for (int i = 0; i < 5; i++) {
        printf("    ");
        for (int j = (i * 20); j < (i * 20) + 20; j++) {
            if (seatLayout[j] == 0) {
                printf("[XX] ");
            }
            else {
                printf("[%02d] ", seatLayout[j]);
            }
        }
        printf("₩n₩n");
    }
    for (int i = 0; i < 100; i++) {
        seatLayout[i] = i + 1;
    }
```

for문을 통해 좌석 번호를 [00]의 형식으로 print하는데, print할 때 위의 inorderSeat 함수에 의해서 0으로 바뀐 좌석은 이미 예약된 노드가 있다는 뜻이므로, 그 좌석은 [XX]로 print합니다. 그 후, 다른 영화의 좌석을 표시하기 위해서 다시 seatLayout을 1~100의 숫자를 가진 배열로 초기화 시킵니다. 각 영화의 좌석정보를 고정적으로 가지고 있는 자료구조를 만드려면 총 27개의 자료구조가 필요합니다. 이는 비효율적이라 판단해, 표시해야할 일이 있을 때마다 tree를 순회하며 확인하는 구조로 구성했습니다.

예매 취소를 할 경우에 요구하는 input은 예약번호 뿐입니다. 예약번호를 유저에게서 받게 되면, 순서대로 day1_t, day2_t, day3_t를 중위순회(inorder)하며, 맞는 예약번호를 가진 node를 탐색합니다. 그리고 찾게 된다면, 유저에게 그 노드가 가지고 있는 예약번호, 일자, 영화명, 시간, 좌석, 결제 수단을 유저에게 확인하게 합니다. 이 때, 결제 수단이 카드였다면, 환불이 카드로 진행될 것임을 고지하고, 현금이었다면 아직 현장에서 현금을 내지 않았으니 환불은 없을 것이라고 고지합니다. 유저가 취소할 예매 정보를 확인하고 삭제하겠다는 선택을 하면, 그 트리에서 해당 node는 rbDelete 함수가 이용돼 삭제됩니다.

예매나 취소를 한 이후에는 다시 초기 메뉴로 돌아갈 지, 프로그램을 종료할 지 선택하게 됩니다. 다시 초기 메뉴로 돌아가 예매나 취소를 이어서 할 수도 있습니다.

레포트의 조건에 따라, 예매를 해서 트리에 node가 추가될 때, 예매 취소를 해서 트리의 node가 삭제될 때마다, 해당 트리의 총 노드 수, root node의 정보, 왼쪽부터 오른쪽까지 leaf node의 정보들, 트리의 높이를 프린트했습니다.

우선, 해당 트리의 총 노드 수는 아래의 countNodes 함수를 이용해 알아냈습니다.

```
void countNodes(Tree* t, Node* n) {
    if (n != t->NIL) {
        countNodes(t, n->left);
        cnt++;
        countNodes(t, n->right);
    }
}
```

Global variable 로서 cnt를 선언한 후, 0 값을 입력했습니다. 그리고 해당 트리를 중위순회(inorder)하며 t->NIL이 아닌 노드를 방문할 경우에는 cnt값을 1씩 증가시켰습니다.

이렇게 된다면, 함수가 끝난 후 총 방문한 노드 수를 알 수 있습니다. 이 함수를 시작할 때 항상 cnt는 0이어야 하고, cnt는 global variable이므로, 위의 함수를 사용한 후에는 항상 cnt를 0으로 재설정 해주었습니다.

트리의 root와 leaf(왼쪽에서 오른쪽으로)의 정보는 아래의 printRootLeaves함수를 이용해 print 했습니다.

```
void inorderLeaves(Tree* t, Node* n) {
    if (n != t->NIL) {
        inorderLeaves(t, n->left);
        if (n->left == t->NIL && n->right == t->NIL) {
            printf("Reservation number : %d₩n", n->rsvNum);
            printf("Date : %d₩n", n->rsvDate);
            printf("Movie : %d₩n", n->rsvMovie);
            printf("Time : %d₩n", n->rsvTime);
            printf("Pay by card : %c₩n₩n", n->payCard);
        }
        inorderLeaves(t, n->right);
    }
}
void printRootLeaves(Tree* t) {
    printf("key value of root : ₩n");
    printf("Reservation number : %d₩n", t->root->rsvNum);
    printf("Date : %d₩n", t->root->rsvDate);
    printf("Movie : %d₩n", t->root->rsvMovie);
    printf("Time : %d₩n", t->root->rsvTime);
    printf("Pay by card : %c₩n", t->root->payCard);
    printf("₩n₩n");
    printf("Leaves (from left to right) : ₩n");
    inorderLeaves(t, t->root);
}
```

printRootLeaves 함수를 보면, 우선 t-> root node에 접근해, 해당 node의 예약번호, 예약일자, 예약 영화, 예약 시간, 결제 수단을 print합니다.

그 후엔 inorderLeaves 함수를 호출합니다.

inorderLeaves 함수를 살펴보면, 해당 tree를 중위순회(inorder)하며, Leaf node란 자식 node가 없는 node이기 때문에 방문한 node의 왼쪽 자식과 오른쪽 자식이 모두 t->NIL 일 경우에, 그 node의 예약번호, 예약일자, 예약영화, 예약시간, 결제수단을 print합니다.

```
int height(Tree* t, Node* n) {
    if (n == t->NIL)
        return 0;
    else {
        int left = height(t, n->left);
        int right = height(t, n->right);
        if (left <= right)
            return right + 1;
        else
            return left + 1;
    }
}
```

Tree의 높이는 위의 height 함수를 이용해 구했습니다. input으로는 해당 tree와 tree의 root node를 넣습니다. 이 때 tree의 root node가 t->NIL이라면 하나의 node도 갖고있지 않은 tree인 것이므로 0을 return합니다. 아닐 경우엔 else문으로 들어갑니다. left에는 계속 left child로 타고 들어가 그 node의 개수, right에는 계속 right child로 타고 들어가 그 node의 개수를 넣습니다. 그 후 그 두 값 중 큰 값에 1(root node)을 더해 return합니다. 재귀함수가 연달아 시행되므로, 꼭 왼쪽으로만 타고 들어간 node의 개수와 오른쪽으로만 타고 들어간 node의 개수만 비교하는 것이 아니라, 계속 비교를 해가며 모든 leaf에서의 left, right 값을 비교해 큰 값을 저장하게 됩니다. 그리고 이 중 결과로 나온 가장 큰 left 나 right의 값이 해당 tree의 높이가 됩니다.

처음 프로그램을 시행할 때, 각 영화 100좌석 중, 30%가 이미 예약된 상태여야 한다는 조건에 따라서 모든 영화의 33 좌석을 무작위로 설정해 예약했습니다.

```
void countNodes(Tree* t, Node* n) {
    if (n != t->NIL) {
        countNodes(t, n->left);
        cnt++;
        countNodes(t, n->right);
    }
}
void randomInitSeat(int* A) {
    A[0] = rand() % 100 + 1;
    for (int i = 0; i < 33; i++) {
        A[i] = rand() % 100 + 1;
        for (int j = 0; j < i; j++) {
            if (A[i] == A[j])
                i--;
        }
    }
}
void randomInitPay(int* P) {
    for (int i = 0; i < 33; i++) {
        P[i] = rand() % 2 + 1;
    }
}
void randomInitBase(int* A, int* P) {
    randomInitSeat(A); //33개의 1~100 난수
    randomInitPay(P); // 33개의 1~2 난수
}
```

우선 randomInitSeat 함수는 A라는 배열에 1~100까지의 난수를 중복되지 않게 생성해 넣습니다. 그리고 randomInitPay 함수는 P라는 배열에 1~2 중 무작위로 생성해 넣습니다. randomInitBase함수를 이용해 이 두 함수를 호출합니다. 즉, 1~100의 겹치지 않는 난수 33개, 1~2의 난수 33개를 생성합니다. 그 후 이 정보를 토대로 예약했습니다.

```
void randomInit(int* A, int* P, Tree* day1_t, Tree* day2_t, Tree* day3_t, int number[], int rsvNum) {
    randomInitBase(A, P);
    for (int d = 0; d < 3; d++) {
        for (int n = 0; n < 3; n++) {
            for (int k = 0; k < 3; k++) {
                for (int i = 0; i < 33; i++) {
                    if (d == 0) {
                        if (P[i] == 1) {
                            rbInsert(day1_t, number[rsvNum], 1, n + 1, k + 1, A[i], 'y');
                            rsvNum++;
                        }
                        else if (P[i] == 2) {
                            rbInsert(day1_t, number[rsvNum], 1, n + 1, k + 1, A[i], 'n');
                            rsvNum++;
                        }
                    }
                    else if (d == 1) {
                        if (P[i] == 1) {
                            rbInsert(day2_t, number[rsvNum], 2, n + 1, k + 1, A[i], 'y');
                            rsvNum++;
                        }
                        else if (P[i] == 2) {
                            rbInsert(day2_t, number[rsvNum], 2, n + 1, k + 1, A[i], 'n');
                            rsvNum++;
                        }
                    }
                    else if (d == 2) {
                        if (P[i] == 1) {
                            rbInsert(day3_t, number[rsvNum], 3, n + 1, k + 1, A[i], 'y');
                            rsvNum++;
                        }
                        else if (P[i] == 2) {
                            rbInsert(day3_t, number[rsvNum], 3, n + 1, k + 1, A[i], 'n');
                            rsvNum++;
                        }
                    }
                }
                randomInitBase(A, P);
            }
        }
    }
```

randomInit함수를 호출하면 우선 randomInitBase 함수를 이용해 위에서 설명한 대로 A와 P 배열을 채워 넣습니다. 그 후, d, n, k, i를 이용해 4중 for 문을 만들었습니다. 변수 d는 날짜, n는 영화, k는 시간, i는 좌석을 가리킵니다. 이렇게 for문을 이용해 3일의 3가지 영화의 3타임에 각각 무작위로 A와 P 배열에 담겨있는 정보를 토대로 33개의 좌석을 무작위의 결제방법과 함께 예약해 각 tree에 297(33 x 9)개의 node를 추가했습니다. 덧붙여 설명하자면, P 배열의 값이 1이면 카드 결제, 2이면 현금 결제로 간주해 예매했습니다.

선택지를 입력하면 원래의 화면이 사라진 후, 다음 화면으로 넘어가기 위해서

system("cls") 함수를 이용했습니다.

2. User Interface Description

```
void UI(Tree* day1_t, Tree* day2_t, Tree* day3_t, int* seatLayout, int number[]) {
    int first, date, times, movie, seat, choice, cancelRsvNum, cancelDate;
    char payCard;
    int rsvNum = 890;

    while (1) {
        rsvNum++;
        printf("**********\n");
        printf("1. Make a reservation\n");
        printf("2. Cancel a reservation\n");
        printf("3. Exit\n");
        printf("**********\n");
        printf("Enter your choice : ");
        scanf("%d", &first);
        system("cls");
        if (first == 1) {
            printf("**********\n");
            printf("1 : 01.12.2020\n");
            printf("2 : 02.12.2020\n");
            printf("3 : 03.12.2020\n");
            printf("**********\n");
            printf("Select the date : ");
            scanf("%d", &date);
            system("cls");
            printf("**********\n");
            printf("1. About time      2. The Avengers       3. Frozen\n");
            printf("1.   12:00              12:00              12:00\n");
            printf("2.   15:00              15:00              15:00\n");
            printf("3.   18:00              18:00              18:00\n");
            printf("**********\n");
            printf("Select the movie number : ");
            scanf("%d", &movie);
            printf("Select the time number : ");
            scanf("%d", &times);
            system("cls");
            if (date == 1) {
                inorderSeat(day1_t, day1_t->root, movie, times, seatLayout);
                printSeatLayout(seatLayout);
```

User Interface는 좌측의 UI() 함수를 이용해 구현해 냈습니다. 우선, 기본적으로 모든 화면의 구성은 printf() 함수를 이용해 안내문 및 선택지들을 콘솔 창에 보여줍니다. 그 후, 유저에게 알맞은 번호를 입력하라고 한 뒤 scanf()함수를 통해 그 값이 어떤 값인지에 따라 각각 다른 if 문으로 들어가 알맞은 기능을 수행하게 구성 되어 있습니다.

```
**********
1. Make a reservation
2. Cancel a reservation
3. Exit
**********
Enter your choice :
```

우 측에 보이는 화면이 프로그램을 처음 실행시키게 되면 나오는 화면입니다. 1을 누르면 예매, 2를 누르면 예매 취소, 3을 누르면 프로그램이 종료됩니다.

첫 번째로, 1을 눌렀을 경우에 대해 설명드리겠습니다.

```
**********
1 : 01.12.2020
2 : 02.12.2020
3 : 03.12.2020
**********
Select the date :
```

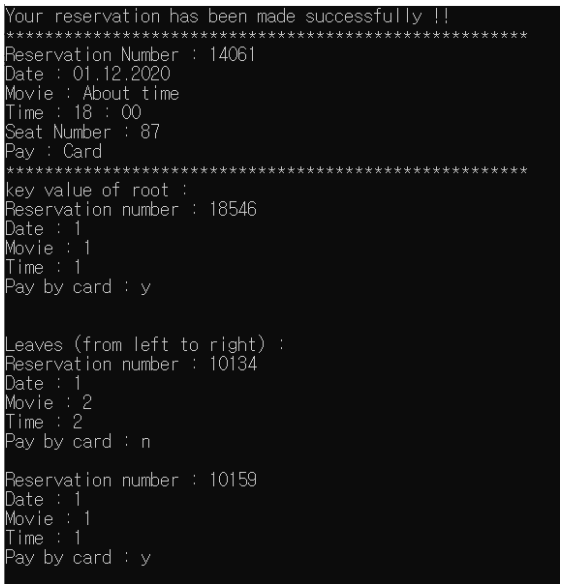
```
**************************************************
1. About time      2. The Avengers       3. Frozen
1.   12:00              12:00              12:00
2.   15:00              15:00              15:00
3.   18:00              18:00              18:00
**************************************************
Select the movie number : 1
Select the time number :
```

상단 좌측과 같이 우선 날짜를 선택하는 화면이나 옵니다. 원하는 날짜의 번호를 입력하면 상단 오른쪽에 보이는 것 같은 화면이 나옵니다. 그 후 원하는 영화의 번호를 입력하면, 아래에 원하는 time number를 선택하라는 안내문구가 나옵니다. 그 시간 number까지 입력하게 되면 아래와 같이 좌석의 정보가 표시 됩니다.

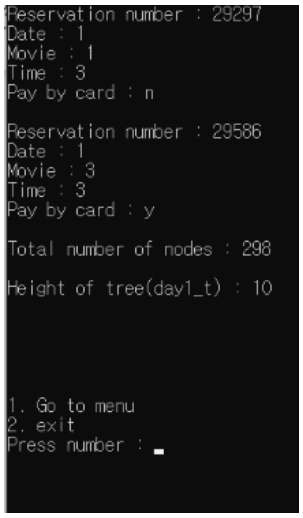
```
                    [SCREEN]

   [01] [02] [XX] [XX] [05] [06] [07] [08] [09] [10] [XX] [12] [13] [XX] [XX] [XX] [17] [18] [19] [20]
   [XX] [XX] [23] [XX] [XX] [26] [27] [28] [29] [XX] [31] [32] [33] [34] [35] [36] [37] [XX] [XX] [40]
   [41] [42] [43] [44] [45] [XX] [47] [48] [49] [50] [XX] [XX] [XX] [54] [55] [56] [57] [58] [59] [XX]
   [61] [62] [63] [64] [65] [XX] [67] [XX] [69] [70] [71] [XX] [73] [XX] [75] [76] [77] [XX] [79] [XX]
   [XX] [XX] [83] [84] [85] [86] [87] [XX] [89] [XX] [91] [XX] [XX] [94] [XX] [96] [XX] [98] [99] [XX]
Please select the seat number : 87
Will you pay by card ? (if yes enter 'y', if no enter 'n') :
```

[XX]라고 표시된 좌석은 이미 예매가 된 좌석이기 때문에 유저는 쓰여있는 숫자 들 중 입력할

수 있습니다. 원하는 좌석 번호를 입력하면 밑에 결제 방법을 입력하라는 문구가 나옵니다.
카드로 결제할 것이면 y를, 현금으로 할 것이면 n을 입력하면 됩니다.



이렇게 모든 정보를 입력하게 되면 예약이 확정되었다는 문구와 함께 예약한 정보가 예약번호와 함께 나타나집니다. 그리고 밑에는 해당 tree의, 현재 예시에서는 01.12.2020에 예약했으므로 day1_t의 root node의 정보들과 leaf들의 값들, 전체 노드의 개수, 트리의 높이가 나타납니다.

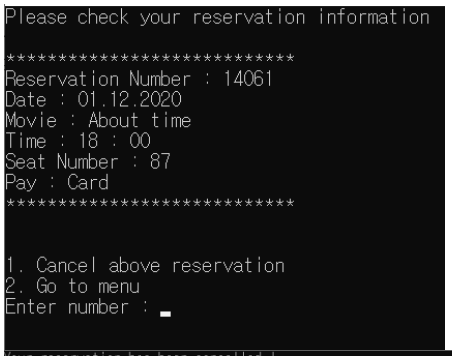


마지막에는 좌측과 같은 초기 메뉴로 돌아갈지 프로그램을 종료할 지 선택할 수 있는 부분이 있습니다. 여기서 1을 입력할 경우, 다시 초기 화면으로 돌아가 이어서 예매를 하거나, 예매 취소를 할 수 있고 2를 입력할 경우 프로그램을 종료시킬 수도 있습니다.

두 번째로, 초기화면에서 2를 입력하는 예매 취소 과정을 설명 드리도록 하겠습니다.

input은 오직 예매 번호입니다.

이 화면에서 유저가 예매했을 시에 배정받은 예매번호를 입력하면 취소할 수 있습니다. 예시로 위에서 에매했던 예매번호 14061의 예매 건을 취소해보기 위해 14061을 입력했습니다.



14061을 입력하게 되면, 좌측과 같이 예매번호 14061의 예매 정보들이 화면에 표시되게 됩니다. 일자, 영화, 시간, 좌석 번호, 결제 방법을 포함하고 있습니다.

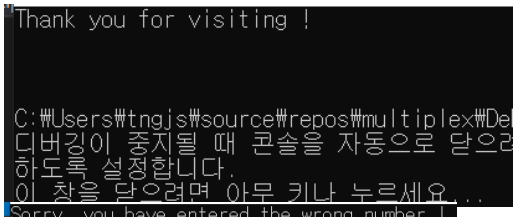
유저가 취소할 예매 정보를 확인하고 맞을 시에는 1을 입력해 위의 예매를 취소할 수 있고, 잘못 되었을 경우에는 2번을 선택해 다시 초기 화면으로 돌아갈 수 있습니다. 1을 입력해 보았습니다.

왼쪽에 보시는 것처럼 예약이 취소되었다는 문구와 함께, 결제 방법이 카드였으므로 환불이 카드를 통해 이루어질 것이라는 문구가 표시됩니다.

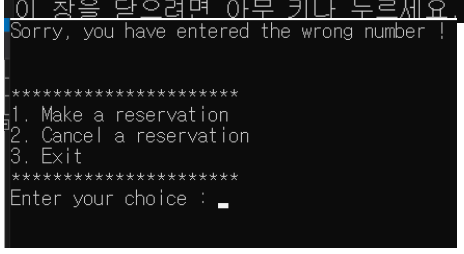
그리고 위의 예매가 확정되었을 경우와 같이 이번엔 rbDelete 함수가 실행되었으므로 tree의 root node와 leaf node들의 정보가 나열됩니다. 이 경우에는 01.12.2020의 예매를 취소했으므로 day1_t의 root와 leaf node들의 정보가 나열되는 것을 보실 수 있습니다.

그리고 역시나 이번에도 나열이 끝나고난 뒤에는 1번을 입력해 초기 메뉴로 돌아갈 수 있고, 2번을 입력해 프로그램을 종료할 수 있습니다.

마지막으로 초기 화면에서 3, 또는 그 외의 숫자를 입력 했을 경우입니다.



3을 입력할 경우에는 감사 문구와 함께 프로그램이 종료됨을 보실 수 있습니다.



이 경우는 1,2,3이 아닌 4를 입력했을 경우입니다. 없는 선택지의 번호를 입력했기 때문에 잘못된 번호를 입력했다는 문구와 함께 다시 처음의 화면이 표시됩니다.

3. Conclusions

처음 프로그램을 구상하면서, 일자, 영화, 시간을 선택할 때 번호 입력이 아닌 실제 문자열을 입력 받는 것이 좋을까 생각해 보았습니다. 하지만 유저 입장에서 영화 명의 띄어쓰기나, 대문자 소문자가 유저마다 다를 수 있고, 시간에서도 24시간 기준으로 작성하는지 12시간 기준으로 작성하는 지 등등, 굉장히 많은 변수들이 존재 했기 때문에 어떤 방법보다 명확한 번호를 입력해 input을 받도록 설계 하였습니다. 참고를 위해 cgv, 메가박스 등등의 홈페이지를 들어가 실제 예약 시스템을 살펴보기도 하였습니다. 그런 사이트들은 모두 클릭을 통해 input을 받는 형식이었습니다. 하지만 제 입장에서는 콘솔이기 때문에 클릭을 통한 input은 불가능하다고 판단했습니다.

또한 난수 생성에서 고려할 사항들이 많았습니다. 처음 했던 생각으로는 첫 번 째 예매는 1번, 두번째는 2번, 이런 식으로 간단한 예약번호를 부여하려고 했습니다. 하지만 이렇게 한다면 현존하는 영화관들의 예약 번호들과의 차이도 심할 뿐더러, 한 자리수 같은 작은 숫자로는 좌석 번호와 헷갈릴 수도 있고, 유일한 번호로 인식되기 힘들 것이라고 생각했습니다. 따라서, 10000~30000 까지의 숫자로 예약번호를 정한 이유는 다섯 자리의 수를 예약번호로 설정하기 위함이었습니다.

세 개의 트리를 사용하는 것에 대한 고민이 많았습니다. 하나의 트리를 사용하게 되면, 예매할 때나 취소할 때 if문을 나누지 않고 그저 인자만 모든 정보를 남아서 건네 준다면 모든 경우가 한번에 다룰 수 있기 때문입니다. 하지만 그렇게 될 경우, insert와 delete한 후에, leaf node들의 정보를 나열하는 과정에서 너무 비효율 적일 것이라 판단했습니다. 또한 하나의 예약 정보를 찾아야할 때, 좌석 정보를 나타내야 할 때, 모든 예약 정보가 들어있는 하나의 tree를 전부 훑어보며 순회해야 합니다. 이 또한 문제라고 생각해 적절한 크기는 일자 하나당 한 개의 tree씩, 즉 전체 정보를 3분의 1로 나눠서 저장하게 되었습니다.