Data Structure Project #1

출제일자: 2019년 09월 11일 (수)

제출일자: 2019년 10월 09일 (수)

학 과: 컴퓨터정보공학부

담당교수: 신영주 교수님

학 번: 2015722068

성 명: 남 윤 창

1. 문제 설명

이번 프로젝트는 Binary Search Tree(BST)를 이용하여 축구 구단 관리 프로그램을 구현하는 것이다. ShootForLog.txt 파일에 선수들의 이적료와 능력치에 대한 정보가 담겨있다. 우선 이는 선수이름, 포지션, 이적료, 능력 순으로 저장 되어있다. 이름은 영어로 작성 되어있다. 포지션의 종류는 Forward, Midfielder, Defender, Goalkeeper로 총 4가지이다. 이적료는 정수 값으로 저장 되어있다. 이 때 같은 포지션 내에는 이적료가 같은 선수는 존재하지 않는다고 가정한다. 능력치는 최저 1부터 최대 99까지의 정수 값을 갖는다. 이적료와 마찬가지로 포지션 내에 능력치가 서로 같은 선수는 존재하지 않는다고 가정한다. 또한, 이적료와 능력치는 비례하지 않는다.

이를 바탕으로 각 포지션별로 선수를 한 명씩 뽑아서 가장 강력한 팀을 만드는 것이 문제이다. 우선 팀의 구성은 4명으로 각 포지션별로 한 명의 선수가 포함된다. 우리는 처음 구단주에게 이적료로 사용할 돈을 받는다. 이 때 4명 선수의 합이 구단주가 제시한 돈의 액수보다 같거나 작아야 한다. 이러한 이적료의 조건하에 4명의 선수 능력치 합이 가장 큰 팀이 가장 강력한 팀으로 정의한다. 만약 능력치가 같은 팀이 생긴다면 이적료가 더 적은 팀이 강력한 팀이 된다. 만약 여기서 이적료까지 같다면 둘은 같은 것으로 취급한다. 이는 총 3개의 단계로 나누어서 이루어진다.

Stage 1. Setup 단계는 ShootForLog.txt에서 선수들의 Data를 읽어와 각 포지션 별로 Tree를 설계한다.

Stage 2. Print Players 단계는 BST에 저장된 선수들을 BST의 특징 중 하나인 In-order search 방식을 사용하여 각 포지션별로 트리의 내용을 모두 출력한다. 공격수, 미드필더, 수비수, 골키퍼 순으로 출력한다.

Stage 3. 구성된 Best Team의 각 소속 선수들을 Tree에서 제거한다. 이후 다시한번 출력한다.

이후 동적할당 한 모든 메모리를 해제하면 된다.

2. 알고리즘 (Pseudo Code & Flow Chart)

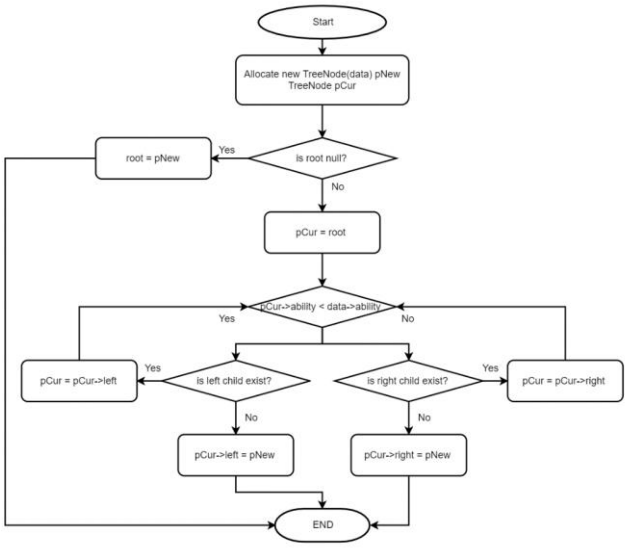
1) Stage 1

ShootForLog.txt 에서 strtok를 사용하여 문자열을 구분한 후 포지션을 통해 각 Tree를 생성하는 코드를 구현하였다. Tree의 Key 값은 선수의 능력치로 주었다. Insert 코드는 다음과 같다.

➔ Pseudo Code (BST Insert)

```
Void insert(SoccerPlayerData & data){
        Allocate new TreeNode node.
```

```
Set Current node pointer
If root is Empty, Set new node as root, END
Else{
        Init cur pointer as root.
        While {
                if data->ability < cur->ability{
                        if left child exists  move down left, END
                        Else      Set cur->left as new node
                }
                Else{
                        If right child exists move down right, END
                        Else      set cur->right as new node
                }
        }
    }
}
```



<Flow chart - insert>

2) Stage 2

Stage 2에서는 미리 만들어 놓은 Tree를 Operator Overloading을 사용하여 출력해주는 단계이다. 이때 능력치가 작은 선수부터 출력해주기 위해 In-order Search 방식을 사용하였다. Recursive와 stack을 사용하는 방식 중 stack을 사용하여 구현하였다.

➔ Pseudo Code (In-order Traversal Print)

If root is not null

Set current node pCur = root stack<TreeNode*> s

Loop

If left child is empty then break

Else{

stack push(pCur)

pCur = pCur->left child

}

Loop

Print player data using operator overloading

If right child is NOT empty {

pCur = pCur->right child then break

else{

if stack is empty then return

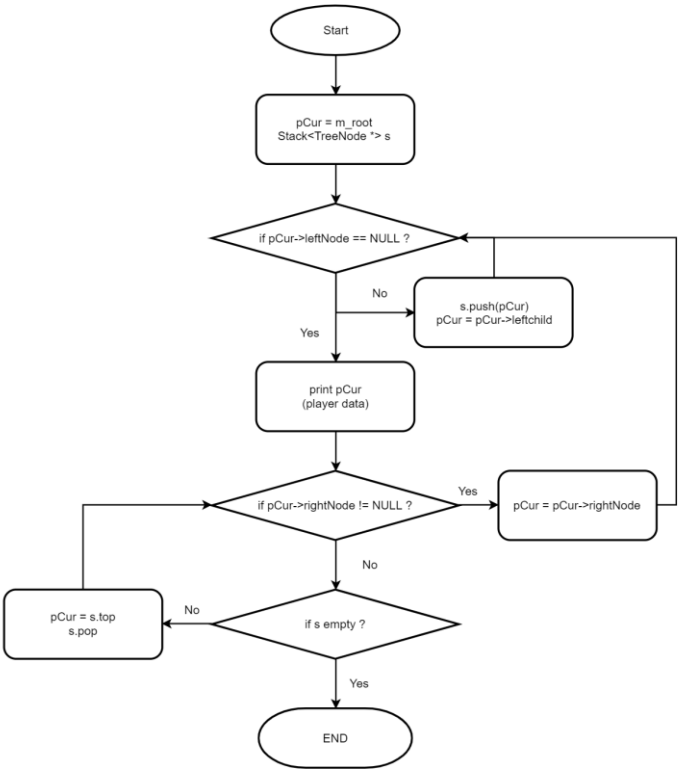
else {

pCur = stack top

stack pop

}

}



<Flow chart - print>

3) Stage 3

Stage 3에서는 주어진 예산안으로 구할 수 있는 가장 최강의 팀을 구하는 것이다. 그 다음 이로 인해 구해진 선수들을 각 트리에서 제거하는 것까지 구현했다. 알고리즘은 다음과 같다.

베스트팀의 코드가 너무 길어 간단히 압축하였다. 원리는 In-order Traversal 이 왼->V->오 순이라면 이를 거꾸로 가장 큰 노드부터 탐색하기 위해 오->V->왼 순으로 탐색하게 하였다. 각 포지션의 가장 큰 노드를 구하여 팀을 꾸린다. 예산을 비교한다. 이후에 현재 포지션의 예산보다 싼 선수중 가장 능력치가 높은 선수를 찾아 각 포지션별로 이동한 팀 3개를 만든다. 이 3팀중 가장 능력치가 높은 팀을 선택하여 다시 반복문을 돌며 예산안이 충족될 때까지 반복하는 알고리즘이다. 이를 위해 flag컨트롤을 해주어 코드가 조금 길어지게 되었다.

아래 수도코드의 Call Function부분이 더 이상 각 포지션에서 좋은 선수를 찾을 수 없을 경우를 예외처리하기 위해 따로 구현한 부분이다. 단순 flag컨트롤이기 때문에 수도 코드에 포함하지 않았다. 전체적인 알고리즘 위주로 작성하였다.

➔ Pseudo Code (GetBestTeam)

```
TransferWindowManager::SoccerTeam TransferWindowManager::getBestTeam()
Initialize with root of each Tree;
Loop 3 Trees each (forward, midfielder, defender)
    Move to right child
    When node is null break;  //move to best ability node
Current Node = top node of each stack;          //forward, midfield, defender
Stack pop;
Loop
    Set team = {current forward, current midfielder, current defender, current gk}
    If budget is not over then break;
    Else
        Set next forward node
        Set next midfielder node
        Set next defender node
        Fw_team = {next forward, cur midfielder, cur defender, cur gk}
        mf_team = {cur forward, next midfielder, cur defender, cur gk}
        df_team = {cur forward, cur midfielder, next defender, cur gk}
        Call Function which compares three team ability and fee
```

If next forward = new best team fwCur = fwNext;

Else if next midfield = new best team mfCur = mfNext;

Else if next defender = new best team dfCur = dfNext;

Deletion(team.fw.abilitiy);

Deletion(team.mf.abilitiy);

Deletion(team.df.abilitiy);

Deletion(team.gk.abilitiy);

➔ Pseudo Code (BST Deletion)

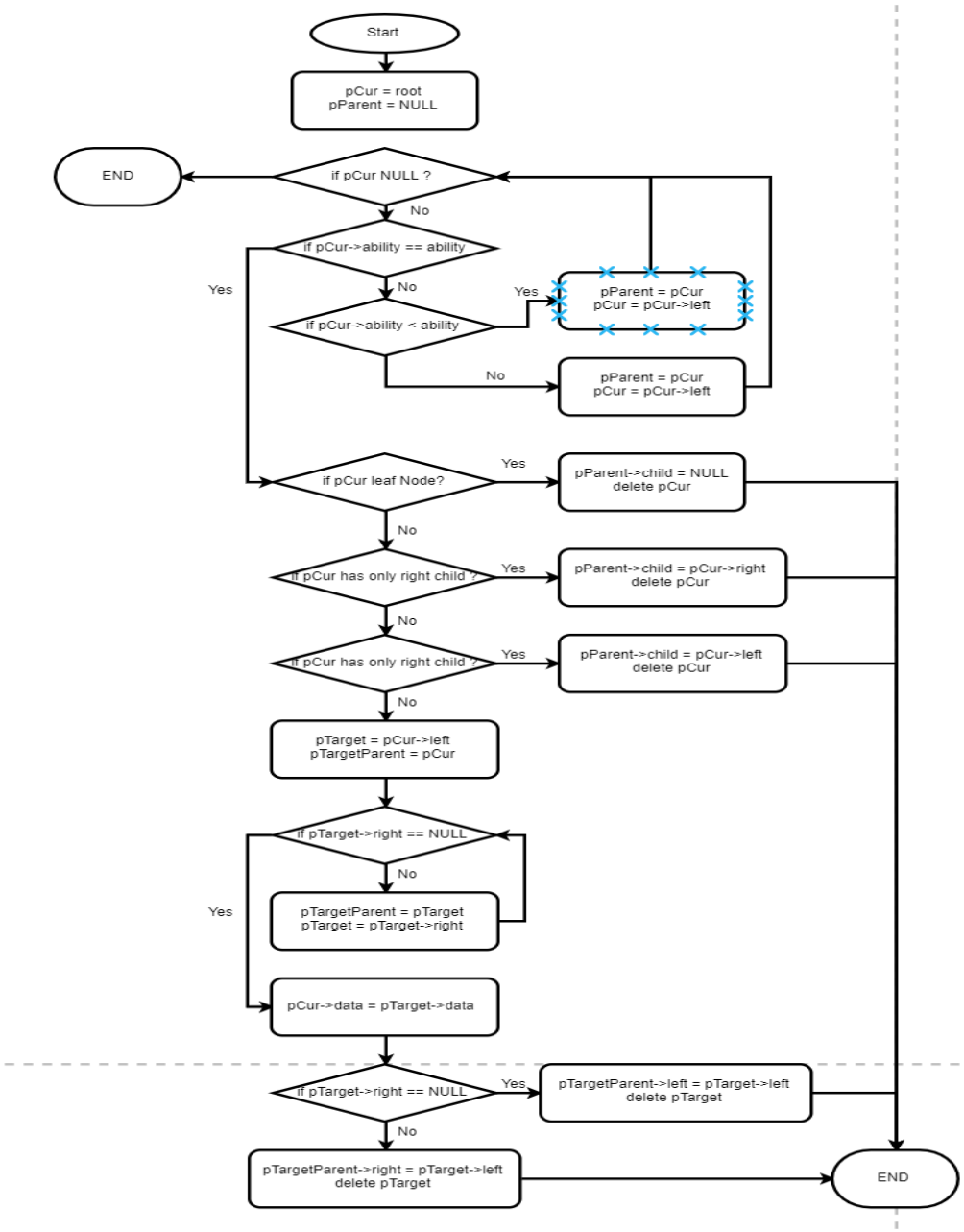
```
Void BinarySearchTree::deletion(int ability)
TreeNode * pCur = root, pParent = NULL
Loop until pCur == NULL or pCur->ability == ability
    pParent = pCur
    if ability is smaller, move left
    else move right
if no matching target, return;
if pCur is Leaf Node
    if pCur is root, root = NULL
    else if pCur is left child of parent, pParent->left = NULL;
    else if pCur is right child of parent, pParent->right = NULL;
    delete pCur and return;
if pCur only has right subtree
    if pCur is root, root = pCur->right;
    else if pCur is left child of parent, pParent->left = pCur->right
    else if pCur is right child of parent, pParent->right = pCur->right
    delete pCur and return;
if pCur only has left subtree
    if pCur is root, root = pCur->left;
    else if pCur is left child of parent, pParent->left = pCur->left
    else if pCur is right child of parent, pParent->right = pCur->left
    delete pCur and return;
Set pTarget as pCur->left, pTargetParent as pCur
Loop until pTarget->right is null
    pTargetParent = pTarget;
    pTarget = pTarget->right;
```

pCur->data = pTarget->data

if pTargetParent is pCur, pTargetParent->left = pTarget->left

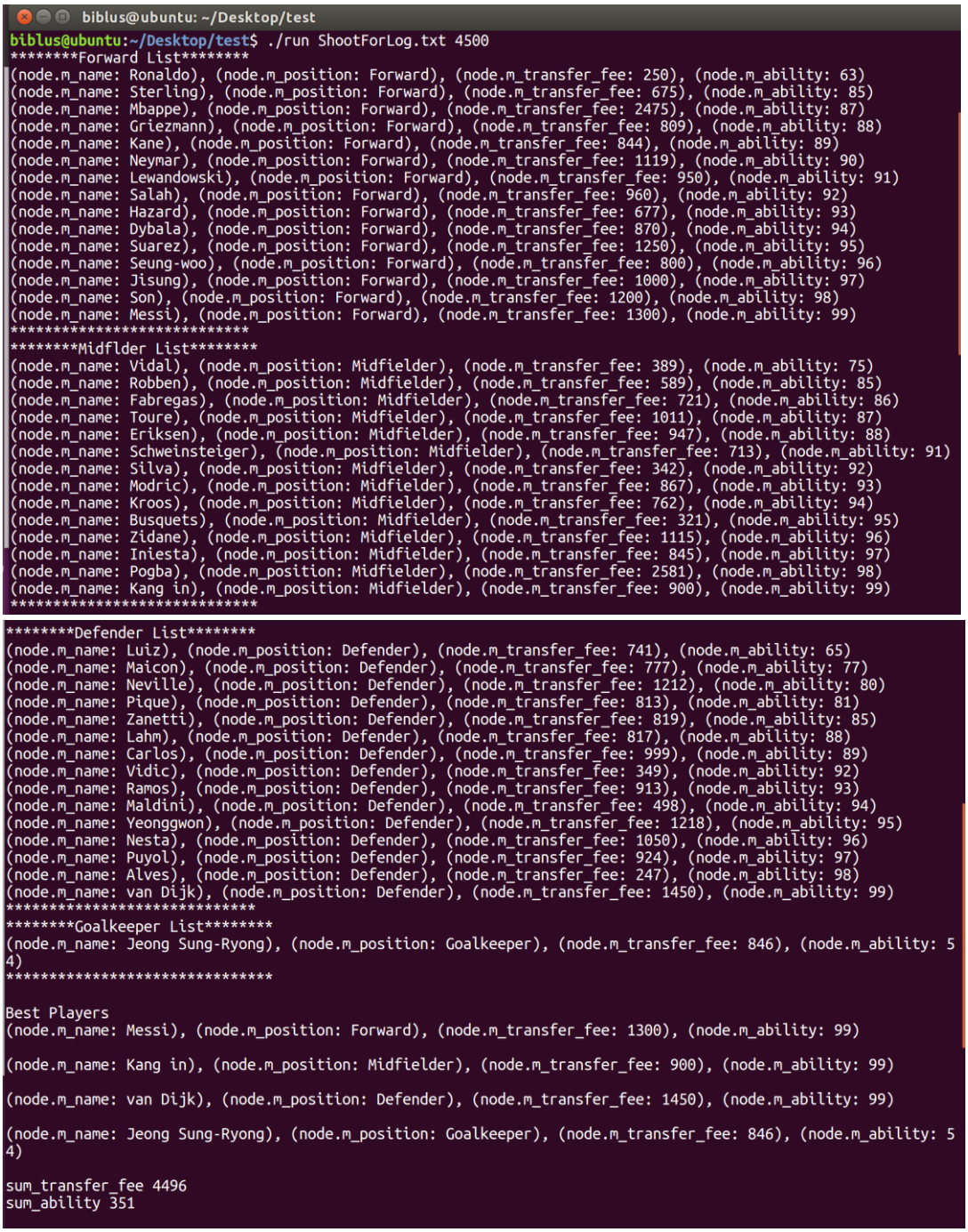
else pTargetParent->right = pTarget->left

delete pTarget

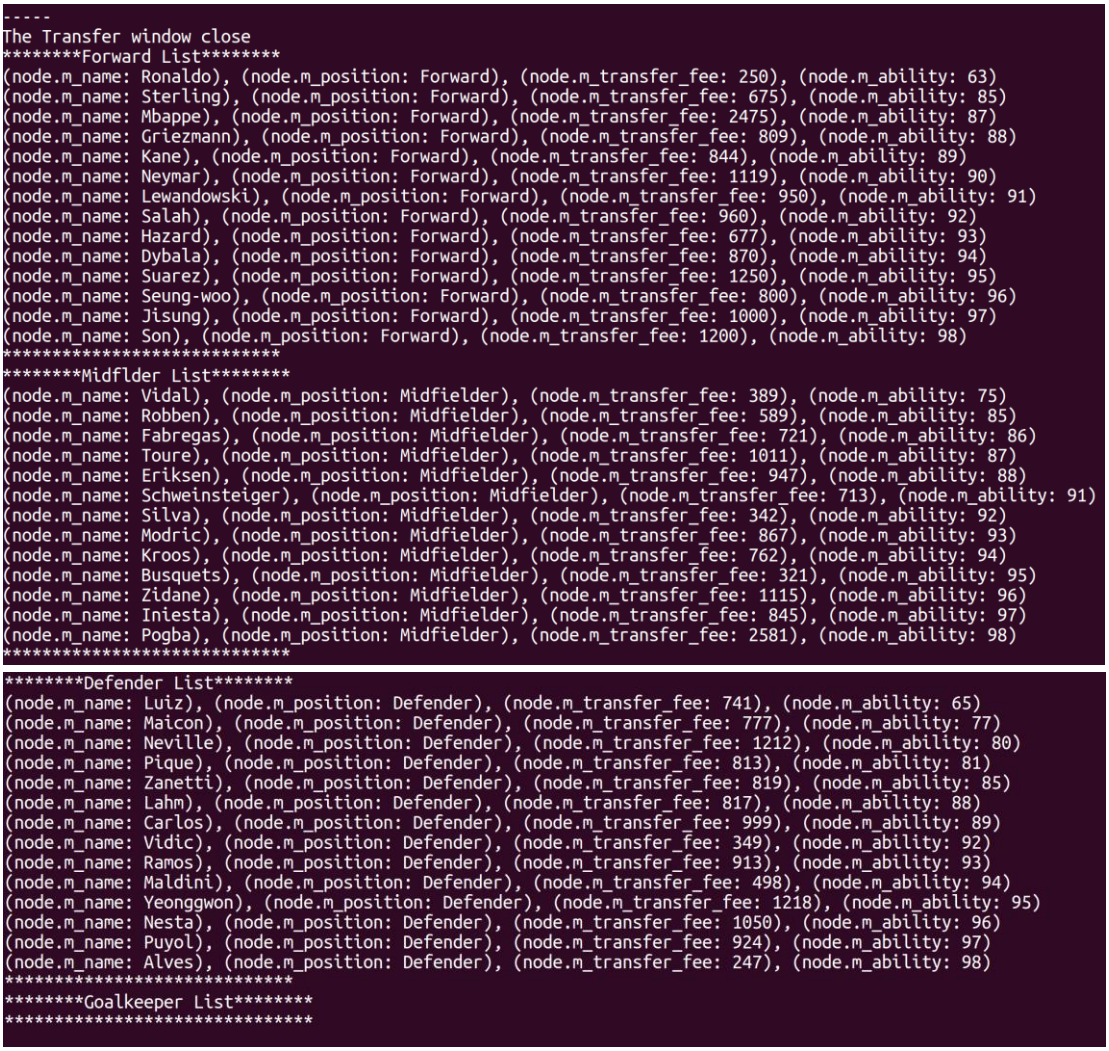


<FlowChart – BST Deletion>

3. 결과(동작) 화면



```
biblus@ubuntu: ~/Desktop/test
biblus@ubuntu:~/Desktop/test$ ./run ShootForLog.txt 4500
********Forward List********
(node.m_name: Ronaldo), (node.m_position: Forward), (node.m_transfer_fee: 250), (node.m_ability: 63)
(node.m_name: Sterling), (node.m_position: Forward), (node.m_transfer_fee: 675), (node.m_ability: 85)
(node.m_name: Mbappe), (node.m_position: Forward), (node.m_transfer_fee: 2475), (node.m_ability: 87)
(node.m_name: Griezmann), (node.m_position: Forward), (node.m_transfer_fee: 809), (node.m_ability: 88)
(node.m_name: Kane), (node.m_position: Forward), (node.m_transfer_fee: 844), (node.m_ability: 89)
(node.m_name: Neymar), (node.m_position: Forward), (node.m_transfer_fee: 1119), (node.m_ability: 90)
(node.m_name: Lewandowski), (node.m_position: Forward), (node.m_transfer_fee: 950), (node.m_ability: 91)
(node.m_name: Salah), (node.m_position: Forward), (node.m_transfer_fee: 960), (node.m_ability: 92)
(node.m_name: Hazard), (node.m_position: Forward), (node.m_transfer_fee: 677), (node.m_ability: 93)
(node.m_name: Dybala), (node.m_position: Forward), (node.m_transfer_fee: 870), (node.m_ability: 94)
(node.m_name: Suarez), (node.m_position: Forward), (node.m_transfer_fee: 1250), (node.m_ability: 95)
(node.m_name: Seung-woo), (node.m_position: Forward), (node.m_transfer_fee: 800), (node.m_ability: 96)
(node.m_name: Jisung), (node.m_position: Forward), (node.m_transfer_fee: 1000), (node.m_ability: 97)
(node.m_name: Son), (node.m_position: Forward), (node.m_transfer_fee: 1200), (node.m_ability: 98)
(node.m_name: Messi), (node.m_position: Forward), (node.m_transfer_fee: 1300), (node.m_ability: 99)
**************************
********Midflder List********
(node.m_name: Vidal), (node.m_position: Midfielder), (node.m_transfer_fee: 389), (node.m_ability: 75)
(node.m_name: Robben), (node.m_position: Midfielder), (node.m_transfer_fee: 589), (node.m_ability: 85)
(node.m_name: Fabregas), (node.m_position: Midfielder), (node.m_transfer_fee: 721), (node.m_ability: 86)
(node.m_name: Toure), (node.m_position: Midfielder), (node.m_transfer_fee: 1011), (node.m_ability: 87)
(node.m_name: Eriksen), (node.m_position: Midfielder), (node.m_transfer_fee: 947), (node.m_ability: 88)
(node.m_name: Schweinsteiger), (node.m_position: Midfielder), (node.m_transfer_fee: 713), (node.m_ability: 91)
(node.m_name: Silva), (node.m_position: Midfielder), (node.m_transfer_fee: 342), (node.m_ability: 92)
(node.m_name: Modric), (node.m_position: Midfielder), (node.m_transfer_fee: 867), (node.m_ability: 93)
(node.m_name: Kroos), (node.m_position: Midfielder), (node.m_transfer_fee: 762), (node.m_ability: 94)
(node.m_name: Busquets), (node.m_position: Midfielder), (node.m_transfer_fee: 321), (node.m_ability: 95)
(node.m_name: Zidane), (node.m_position: Midfielder), (node.m_transfer_fee: 1115), (node.m_ability: 96)
(node.m_name: Iniesta), (node.m_position: Midfielder), (node.m_transfer_fee: 845), (node.m_ability: 97)
(node.m_name: Pogba), (node.m_position: Midfielder), (node.m_transfer_fee: 2581), (node.m_ability: 98)
(node.m_name: Kang in), (node.m_position: Midfielder), (node.m_transfer_fee: 900), (node.m_ability: 99)
**************************
********Defender List********
(node.m_name: Luiz), (node.m_position: Defender), (node.m_transfer_fee: 741), (node.m_ability: 65)
(node.m_name: Maicon), (node.m_position: Defender), (node.m_transfer_fee: 777), (node.m_ability: 77)
(node.m_name: Neville), (node.m_position: Defender), (node.m_transfer_fee: 1212), (node.m_ability: 80)
(node.m_name: Pique), (node.m_position: Defender), (node.m_transfer_fee: 813), (node.m_ability: 81)
(node.m_name: Zanetti), (node.m_position: Defender), (node.m_transfer_fee: 819), (node.m_ability: 85)
(node.m_name: Lahm), (node.m_position: Defender), (node.m_transfer_fee: 817), (node.m_ability: 88)
(node.m_name: Carlos), (node.m_position: Defender), (node.m_transfer_fee: 999), (node.m_ability: 89)
(node.m_name: Vidic), (node.m_position: Defender), (node.m_transfer_fee: 349), (node.m_ability: 92)
(node.m_name: Ramos), (node.m_position: Defender), (node.m_transfer_fee: 913), (node.m_ability: 93)
(node.m_name: Maldini), (node.m_position: Defender), (node.m_transfer_fee: 498), (node.m_ability: 94)
(node.m_name: Yeonggwon), (node.m_position: Defender), (node.m_transfer_fee: 1218), (node.m_ability: 95)
(node.m_name: Nesta), (node.m_position: Defender), (node.m_transfer_fee: 1050), (node.m_ability: 96)
(node.m_name: Puyol), (node.m_position: Defender), (node.m_transfer_fee: 924), (node.m_ability: 97)
(node.m_name: Alves), (node.m_position: Defender), (node.m_transfer_fee: 247), (node.m_ability: 98)
(node.m_name: van Dijk), (node.m_position: Defender), (node.m_transfer_fee: 1450), (node.m_ability: 99)
**************************
********Goalkeeper List********
(node.m_name: Jeong Sung-Ryong), (node.m_position: Goalkeeper), (node.m_transfer_fee: 846), (node.m_ability: 5
4)
*****************************

Best Players
(node.m_name: Messi), (node.m_position: Forward), (node.m_transfer_fee: 1300), (node.m_ability: 99)

(node.m_name: Kang in), (node.m_position: Midfielder), (node.m_transfer_fee: 900), (node.m_ability: 99)

(node.m_name: van Dijk), (node.m_position: Defender), (node.m_transfer_fee: 1450), (node.m_ability: 99)

(node.m_name: Jeong Sung-Ryong), (node.m_position: Goalkeeper), (node.m_transfer_fee: 846), (node.m_ability: 5
4)

sum_transfer_fee 4496
sum_ability 351
```



액수를 4500원을 주었을 경우 모두 99능력치의 선수들로 이루어진 팀이 구성되고 이를 Stage3에서 deletion을 한 이루 한번 더 출력함을 확인했다.

4. 고찰

이번 과제는 Binary Search Tree를 활용하여 내가 축구 구단 이적료 관리 프로그램을 구현하는 것이었다. BST의 Insert Delete등은 난이도가 쉬운 편이었지만 Best Team을 구하는 알고리즘을 생각하는데 오랜 시간을 투자했다. InOrder Traversal을 가장 큰 노드부터 역방향으로 탐색하며 하나씩 비교하고 액수가 충족되면 멈추는 방식으로 구현하였는데 이를 막상 구현하려고 하니 너무 복잡하고 어려웠다. 직접 손으로 flowchart를 그리며 구현하였는데 if else문이 너무 자잘하게 많이 나와서 어려웠다. 더 좋은 방식이 있을 것 같기도 한데 아직 많은 것을 배우지 않아 어려웠다. Queue를 사용하여 구현하면 더욱 좋고 직관적인 코드를 구현할 수 있을 것 같은 느낌이 든다. 그리고 Deletion을 구현할 때 처음에는 손에 잡히는 대로 코딩하였다. 그러나 수업을 듣고 나서 내 코드가 스파게티 코드임을 깨닫고 다시 전부 지우고 구현하였다. 이때 버전관리를 위하여 깃을 사용하여 혹시 모를

코드의 분실도 고려해주었다. 마지막으로 리눅스를 처음 사용하였는데 처음엔 VMware자체 오류가 너무 많이 발생하여 Linux로 확인하는데 어려움이 있었다. 그래서 VMWare를 15버전으로 바꾸고 나니 파일을 옮기는 데는 에러가 생기지 않았고 Segmentation Fault도 모두 고칠 수 있었다. 처음 Segmentation Fault는 Window에서 구현하기 위해 임의로 선언해준 file_dir, budget변수를 고치지 않은 채로 linux에서 compile했었고, 두번째 오류는 strcpy등의 스트링 헤더를 사용하는데 스트링 헤더를 선언하지 않았었다. 다 구현하고 보니 막상 어려운 문제는 아니었지만 연산자 오버로딩과 friend의 사용이 처음에 당황하게 하는 부분이었고 취약한 C++ 문법 개념을 다시한번 복습하고 갈 수 있는 좋은 기회였다고 생각한다.