데이터구조설계(실습-금) 보고서

제출일: 2019.10.09(수)

담당교수:신영주 교수님

학과:컴퓨터정보공학부

학번:2018202074

이름:김상우

문제 설명

제시 조건)

기존의 제공된 스켈레톤 코드를 지키며 제작해야 한다. 이 경우 사용된 함수나 변수들을 변경이 허용될 경우를 제외하고 유지할 것을 의미한다. 단 멤버 변수나 함수 등의 추가는 허용한다. STL 사용은 자유롭다.

프로그램 종료 시 Segmentation Fault가 뜨지 않아야 하며, 이전에 동적할당 된 모든 데이터의 메모리는 해제되어야 한다.

-프로그램의 작동 과정

1)'ShootForLog.txt'와 budget을 명령 인수로 받고 내용들을 바탕으로 선수들의 내용을 읽어낸다.

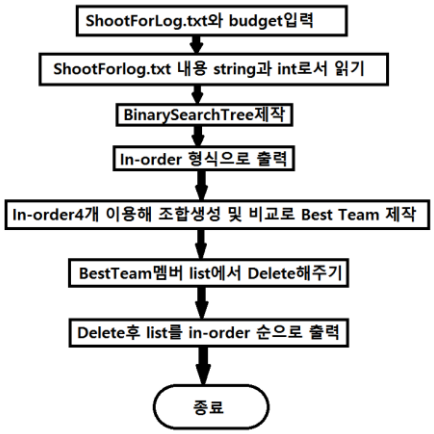
2)축구선수들의 ability를 기반으로 BinarySearchTree를 작성한다. abiltiy가 적을 수록 좌측 노드에 클수록 우측노드에 저장하게 된다.

3)이 BinarySearchTree를 이용해 중위 순회를 하면서 내용들을 출력한다. 이는 연산자 오버로딩을 통해 <<가 다시금 출력형식이 변경되게 한다.

4)선수들의 정보 중 ability와 입력된 budget로 BestTeam을 찾고 그 BestTeam을 출력한다.

5)또한 그 BestTeam에 속한 멤버들을 list에서 제거한다.(이때 제거는 기존에 제시된 특정함수에서 이루어지며 삭제 방식은 배경지식의 설명을 따른다.)

6)BestTeam이 제거된 list를 In-order순으로 출력한다. 끝에서는 메모리 해제를 진행해 준다.



해결을 위한 전체 flow chart)

넣는 방식: 위 그림처럼 root(A(20))을 시작으로 더 작은 값을 왼쪽 더 큰 값을 오른쪽으로 넣는 식으로 진행되며 최종적으로 들어간 값을 그 상태의 leaf가 된다.

빼는 방식: 이는 node의 상태에 따라서 다르게 사용된다.

1)leaf노트를 빼는 경우: 해당 노드를 없앤다.

2)왼쪽 자식 노드만이 존래할 경우: delete한 노드를 왼쪽 자식 노드로 대체한다.

3)오른쪽 자식 노드만이 존재할 경우:delete한 노드를 오른쪽 자식 노드로 대체한다.

4)양쪽 자식 노드가 모두 존재할 경우: delete한 노드의 왼쪽 서브트리 중 가장 큰 값으로 대체한다.

스테이지별 알고리즘 설명

설명에 앞서 각 파일에서는 적합한 friend를 사용하고 있다. Friend는 다른 클래스간의 private와 protected 멤버를 접근할 수 있게 해주는 지시어로서(함수가 아니다.) 서로의 함수를 사용하기 위해서 이 코드에 사용되었다.

간단한 예시로 class a{private: string name; friend class Friend1;};

class b{public:void set_name(a&f, string s){f.name=s;}}라 명시한다면

이후 main함수에서 a f1; b f2; f2.set_name(f1, "name1");이라 명시할 경우 f1에 name1이라는 이름이 name에 저장된다. 이를 숙지하고 코드를 짜야할 것이다.

Stage 1. Setup)

DS_Project1.cpp{

TransferWindowManager transfer_window_manager(argv[1], atoi(argv[2])that read txt and budget

TransferWindowManager.cpp)(with txt file and budget){

Put budget value in m_budget (M_budget=budget)

Open file(ifstream in) and make string

String line, i_name, i_position, power, money

Int i_power, i_money;

While(getline(in, line)){

i=0;

int nPos=0; to check size of vector and type of information

vector<string> tokens;

while(nPos=line.find(",",0)!=string::npos){

tokens.push_back(line.substr(0,nPos))

line=line.substr(nPos+2);

```
                    }
                    Tokens.push_back(line);
                    For(string token:tokens){
                        Put token in i_name, i_position, money, power by value of i
                        Use atoi about power(money) and put them in i_power(i_money)
                        i++;
                    }
                    SoccerPlayerData a(i_name, i_position, i_money, i_power);
                    Check i_position by if and compare function with string Forward, Midfielder, Defender, Goalkeeper
                    If(use function compare to i_position and stings"Forward, Midfielder, Defender, Goalkeeper"(use else))
                        (find appropriate binarysearch list(fwBST, mfBST, dfBST, gkBST) and use insert in that)
                        BinarySearchTree::insert(SoccerPlayer& data){
                        Tree*p=m_root, *pp = NULL;
                        While(p){
                                If(data's ability<m_root(p)'s ability) p=p->m_left; #p go in p's left
                                Else if(data's ability>m_root(p)'s ability) p=p->m_right #p go in p's right;
                                Else{put p's m_name, m_position, m_transfer_fee in data' elements} and return;
                        }
                        P=new TreeNode(data);
                        If(m_root != NULL){
                                If(data.m_ability < pp->m_data.m_ability) pp->m_left = p; #make p's up side
                                Else pp->m_right = p;
                        }
                        Else{m_root = p;};
                        }
                    }
            }
    }
  }
}
```

파일 입출력을 위해 명령인수로 txt파일을 받고 있다.(이때 사용 가능 금액도 불러온다.) 이후 이 금액은 다른 변수에 넣어준다. txt파일 같은 경우에는 'ifstream'과 'getline'을 통해 텍스트 파일을 열고 한 줄씩 읽고 있다. 또한 string형태의 벡터를 선언하고 문자열에서의 위치를 반환하는 find와 substr 함수를 이용해 ,를 기준으로 정보를 나누어 벡터인 line에 넣어주고 있다. 이 과정에서 각 줄 별로 몇 번째 string이 들어왔는지 확인이 가능하며 이를 통해 적합한 변수들에 값을 저장했다. Fee와 ablility의 경우에는 이후의 비교를 위해 atoi함수를 이용, int의 형태로 바꾸어준다.

이후 저장한 position을 확인하고 이 값을 compare을 통해 지정된 문자열과 비교한다. 이를 통해 확인된 player의 포지션에 따라 알맞은 BST에 Player data를 추가한다. 이 과정에서 Ability또한 비교하게 되는데, 이 비교를 통해 node상의 위치를 파악한다. 이는 insert함수에서 처리함을 위 의사코드에서 확인할 수 있다.

insert함수는 insert할 값을 저장할지 확인할 Treenode와 저장될 노드의 상위 Treenode를 지정한다. 위에서 확인할 수 있듯이 노드가 존재하는 이상 노드의 위치는 계속 하위 노드로 내려가며 확인중인 Treenode의 ability보다 낮을 시 왼쪽으로 높을 시 오른쪽을 향한다.

Stage 2. Print Player)

```
DS_Project1.cpp) {

   Print   transfer_window_manager and change line

   TransferWindowManager.cpp){
   std::ostream& operator<<(std::ostream& os, const TransferWindowManager& manager){
   put "********Forward List********" in os and change line
   put manager.fwBST in os                                    #print BinarySearchTree about forward
      BinarySearchTree.h){
       const void Inorder(std::ostream &out, const TreeNode*tree) const {
            if tree isn't nullptr) {
               do self-function about out and tree->left
               put tree in out and change line
               TreeNode,h){
                  friend std::ostream& operator<<(std::ostream& os, const TreeNode* node){
                     if node isn't NULL,
                        put node->m_data in os
                        SoccerPlayerData.h){
                           friend std::ostream& operator<<(std::ostream& os, const SoccerPlayerData& node) {
                              put node's m_name ,m_position ,m_transfer_fee ,m_ability in os with comment
                              return os;
                              }
                        }
                     return os;
                  }
               }
               do self-function about out and tree->right
               }
            }
         friend std::ostream& operator<<(std::ostream& os, const BinarySearchTree& tree) {
            do self-function about out and tree->right
            return os;
    }
   put manager.mfBST, manager.dfBST, manager.gkBST in os with comment and line change
   return os;
}
}
```

-friend std::ostream& operator<<(std::ostream& os, const)

사용자 정의 클래스 사용시 해당 클래스에 대하여 연산자를 멤버함수 또는 함수로 정의하여 실질적인 내용을 바꾸는 방법인 연산자 오버로딩을 위에선 <<에 사용하고 있다

위 함수는 각 결과들을 출력하는 함수로서 사용되어 위 Step에서 반복적으로 쓰이고 있다.
const뒤에 필요한 변수를 넣고 이를 적절한 형식으로 os에 입력하고 return하는 형식으로 진행되고 있는 것 또한 확인할 수 있다.
대표적으로 SoccerPlayerData.h에서는 const SoccerPlayerData& node가 두번째 변수로 들어가며

```
os << "(node.m_name: " << node.m_name << "), "
                        << "(node.m_position: " << node.m_position << "), "
                        << "(node.m_transfer_fee: " << node.m_transfer_fee << "), "
                        << "(node.m_ability: " << node.m_ability << ")";
        return os;
    }
```

형식으로 os에 node에 name, position, transfer_fee, ability가 형식을 위한 입력들 사이에 들어감을 알 수 있다.SoccerPlayerData.h를 제외하고 다른 파일에서는 출력이 아닌 다른 파일들의 이 함수를 불러오고 있는 것도 확인이 가능하다. 이는 string과 int로 출력이 가능한 SoccerPlayer.h의 값을 불러오기 위한 과정이라고도 볼 수 있다. 이로서 <<를 통해 하위 함수들도 불러올 수 있음을 알 수 있다.

다만 BinarySearchTree.h의 경우 tree.Inorder라고 하는 파일로 되어있다.

이는 const void Inorder함수에서 Inorder함수를 통해 treenode를 확인하고 이가 nullpointer가 아닐 경우 왼쪽 노드에 대한 Inorder을 진행을 하고, out<< tree << std::endl; (즉, 다른 함수와 같이 os에 값을 넣어주고)를 진행하며 이후 오른쪽 노드에 대한 Inorder를 진행하는 함수이다.

결과적으로 값을 넣어주는 방식은 비슷하지만 In-order 순으로 출력하기 위해 중위순회를 진행하는 다른 함수를 제작한 것이다.

결과적으로 큰 정보에서 작은 정보 순으로 출력이 가능한 시점까지 내려가 출력이 가능하도록 만든 것이다.

Stage 3. Search the Best Team & Delete Players of Best team from the fwBST, mfBST, dfBST and gkBST)

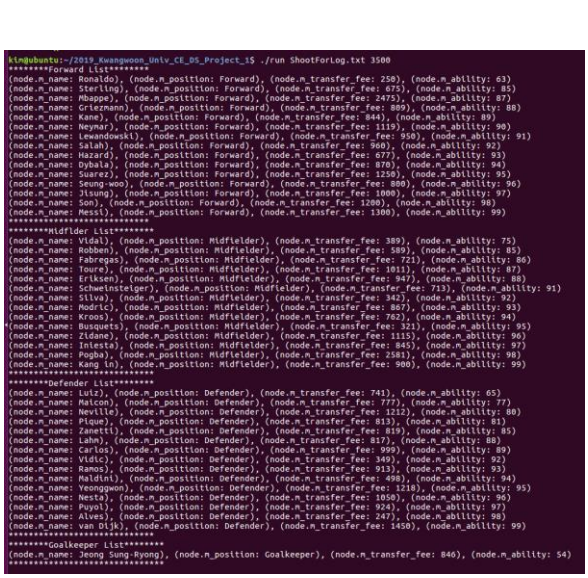
```
DS_Project1.cpp) {

  TransferWindowManager::SoccerTeam team = transfer_window_manager.getBestTeam();
    TransferWindowManager::SoccerTeam TransferWindowManager::getBestTeam(){
      SoccerTeam best_team;
      TreeNode * cur_fwBST = fwBST.m_root;
      TreeNode * cur_mfBST = mfBST.m_root;
      TreeNode * cur_dfBST = dfBST.m_root;
      TreeNode * cur_gkBST = gkBST.m_root;
      Int best = 0; # initial variable to compare add of value of ability
      inorder(cur_fwBST, cur_mfBST, cur_dfBST, cur_gkBST, &best);
        void TransferWindowManager::inorder(TreeNode* a, TreeNode*b, TreeNode*c, TreeNode*d, int * best) {
          if (a is exist) {   do self-function about a->left and same input(recursive)
                    *four loop about treenode * b, c, d for check every team between inorder and inorder4
                    void TransferWindowManager::inorder4(same with function inorder's input) {
                      if (d is exist) {
```

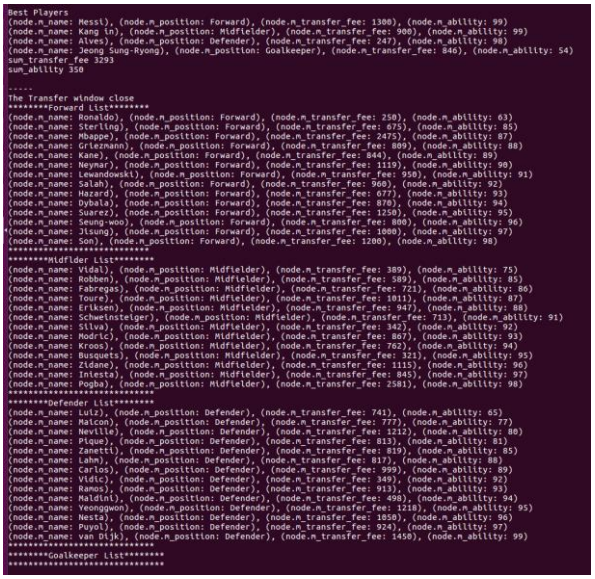
```
                    do self-function about d->left and same input(recursive)
                    cmpbud(&(a->m_data), &(b->m_data), &(c->m_data), &(d->m_data), best);
                        void TransferWindowManager::cmpbud(same with function inorder's input) {
                          if (m_budget >= (add of a,b,c,d's fee)){
                                if (*best < add of a,b,c,d's ability){
                                        *best = add of a,b,c,d's ability;
                                        this->fw2 = *a; this->mf2 = *b;
                                        this->df2 = *c; this->gk2 = *d;
                                }
                          }
                        }
                    do self-function about d->right and same input(recursive)

                }
                do self-function about a->right and same input
    }
  }
SoccerTeam best_team2(fw2,mf2,df2,gk2);#put value in temporary variable
best_team = best_team2;                    #put same value in real variable
fwBST.deletion(best_team.fw.m_ability);
    BinarySearchTree.cpp)void BinarySearchTree::deletion(int ability) {
      TreeNode* checknode = m_root, * delnode = 0;
      while (checknode is exist and ability is not same with checknode->m_data.m_ability) {
        delnode is same with checknode;
        if (checknode->m_data.m_ability) is bigger than ability {
                checknode go its left
        }
        else if (checknode->m_data.m_ability) is smaller than ability {
                checknode go its right
        }
      }
      if (checknode is not exist)return;
      if checknode don't have left and right node{
        if        delnode is not exist, m_root = 0;
        else if delnode->m_left is same with checknode, delnode->m_left = 0;
        else     delnode->m_right = 0;
        delete checknode about memory; return;
      }
      else if checknode have only left node {
        if        delnode is not exist,          m_root = checknode->m_right;
        else if delnode->m_left is same with   checknode, delnode->m_left=checknode->m_right;
        else      delnode->m_right = checknode->m_right;
        delete   checknode about memory return;
      }
      else if checknode have only right node {
        if       delnode is not exist,              m_root = checknode->m_left;
        else if delnode->m_left is same with checknode, delnode->m_left = checknode->m_left;
        else     delnode->m_right = checknode->m_left;
        delete   checknode about memory; return;
```

```
                }
                else if checknode have both node{
                   TreeNode*prevprev = checknode, *prev = checknode->m_left, *curr = checknode->m_left->m_right;
                   while (curr) {
                           prevprev = prev; prev = curr; curr = curr->m_right;
                   }
                }
             }
       Use deletion to mfBST. dfBST, gkBST
        return best_team;
  }
  Print "Best Players"   and change line
  Print team and change line based under function
     std::ostream& operator<<(std::ostream& os, const TransferWindowManager::SoccerTeam& team)
     {
        Put team's fw, mf, df, gk, sum_transfer_fee, sum_ability in os with comment and change line
        return os;
     }
  print "-----" and change line
  print "The Transfer window close" and change line
  print transfer_window_manager like step2's sudo code and change line
return 0;

  BinarySearchTree.cpp)
  BinarySearchTree::~BinarySearchTree() {
           Call function FreeTree(m_root);
  }
  void BinarySearchTree::FreeTree(TreeNode* a) {
     if a is exist) {
           if a have left node) {
do self function about a's LeftNode;
           }
           if a have right node) {
              do self function about a's RightNode;
           }
              Delete a about memory
           }
     }

}
```

Best Team을 찾아내기 위해 Inorder, 즉 중위순회를 통해 모든 soccerplayer의 조합을 확인하고 이들의 가격이 최대 사용 금액 이하이며, 이전 조합의 능력치의 합보다 높은지 계속해서 확인한다. 또한 deletion함수를 이용해 계속해서 삭제를 진행해준다.

결과:





위의 사진에서 확인 가능하듯이 github에 올라와 있는 결과와 같은 결과물을 내는 것을 확인할 수 있다. 이는 다른 예시 결과값에서도 잘 작동하는 것을 볼 수 있다.

./run ShootForLog.txt 2000에서의 BestTeam. 예시와 같게 나오는 것을 확인할 수 있다.

```
Best Players
(node.m_name: Ronaldo), (node.m_position: Forward), (node.m_transfer_fee: 250), (node.m_ability: 63)
(node.m_name: Busquets), (node.m_position: Midfielder), (node.m_transfer_fee: 321), (node.m_ability: 95)
(node.m_name: Alves), (node.m_position: Defender), (node.m_transfer_fee: 247), (node.m_ability: 98)
(node.m_name: Jeong Sung-Ryong), (node.m_position: Goalkeeper), (node.m_transfer_fee: 846), (node.m_ability: 54)
sum_transfer_fee 1664
sum_ability 310
```

고찰)

우선 friend의 사용방법이 가장 인상 깊게 다가왔다. 다른 함수의 요소들을 쓰기 위해서는 다른 요소를 추가하는 등의 불편함이 있었지만 friend의 예시 등을 바탕으로 더 간결하고 효율적인 코드를 짜는 것이 가능했다. 아쉬운 점이 있다면 중위 순회를 할 때 stack으로도 짤 수 있었을 것 같다는 것이다. 위 코드에서는 stack을 이용하지 않고 구현하였지만 언젠가 stack으로도 구현해 볼 생각이다.