
자료구조 설계과제 보고서

[자료구조 과목 관리 프로그램]

010-8578-7628

12131489 김영훈

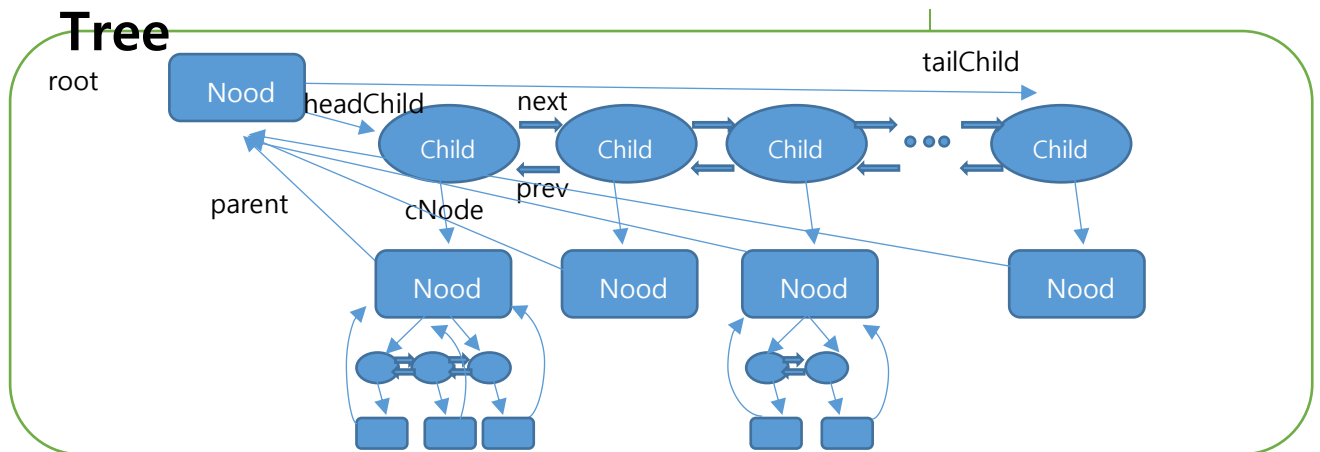
12131489@inha.edu

1. 개요

이 프로그램의 설계의 목적은 수강생 수의 증가에 따라 구성원들을 분반별로 분류하여 효율적인 관리를 체계를 구성함에 있다. 이 프로그램을 설계하기 위해 트리와 링크드리스트의 개념과 구조에 대한 전반적인 이해가 필요하다. 이 프로그램은 운영체제 Windows10에서 개발도구 Microsoft Visual Studio Community 2015를 이용하여 C++ 언어로 개발하였다.

2. 필요한 자료구조 및 기능

1. Tree를 구현하기 위해 사용한 세부 자료구조



Tree	Node	Child
<pre> class Tree { public: Node* root; void childList(Node* newNode); // 자식 링크드리스트 삽입 void nodeInsert(string name, int group, string duty, int id); void groupPrint(int group); // 전체 목록이나 분반의 목록을 받o void Print(Node* printNode, Child* printChild); // groupPrinto void Delete(string name); Tree() { root = NULL; } }; </pre>	<pre> class Node { public: Node* parent; Child* headChild; // 노드의 자식 Child* tailChild; // 노드의 자식 info info; int childNum; // 자식의 개수 Node() { parent = NULL; headChild = NULL; tailChild = NULL; childNum = 0; } }; struct info { //노드의 정보 string name; int group; string duty; int id; }; </pre>	<pre> class Child { // 노드의 자식 링크드리스트 public: Child* next; Child* prev; Node* cNode; // 자식노드를 가르킴 Child() { next = NULL; prev = NULL; cNode = NULL; } }; </pre>

Tree를 구현하기 위해 Tree, Nood, Child 클래스를 만들었다. Tree 클래스는 트리 전체를 나타내는 것이고 root를 포함하고 있다. 또한 각 기능을 포함한 함수들이 포함 되어있다. Nood 클래스는 개인의 정보(이름, 분반번호, 직책, ID) 담고 있는 노드이며, Child의 head와 tail을 담고있다. Child 클래스는 하나의 부모와 여러 자식들이 parent-child 관계를 만들어 주기 위한 클래스이다. 형제 노드들과 double linked list 형태로 서로를 가리키도록 구현하였다.

2. 프로그램들의 기능 설명

이 프로그램은 출력, 삽입, 삭제 3가지의 기능을 포함하고 있다. 출력 기능은 분반번호 *s*를 입력하면 해당 분반의 강의교수와 수강생의 목록을 출력한다. 0을 입력한 경우 책임교수를 포함한 트리 전체의 노드들을 출력한다. 삽입기능은 이름, 분반번호, 직책, ID를 받아 새로운 노드를 생성하여 트리에 추가한다. 삭제기능은 이름을 입력 받아 해당 이름에 해당되는 인원을 트리에서 삭제한다.

3. 기능별 알고리즘 명세

1. 삽입

void Tree::nodeInsert(string name, int group, string duty, int id)

```
void Tree::nodeInsert(string name, int group, string duty, int id) {
    Node* newNode = new Node;
    newNode->info = { name, group, duty, id };
    if (duty == "책임교수" && group == 0) { // 책임교수이면 루트(가장 상위 노드)가 된다.
        root = newNode;
    }
    else if (duty == "강의교수" && group != 0) { // 강의교수일 경우 루트를 부모로 하고 루트의 자식 링크드리스트에 추가시킨다.
        newNode->parent = root;
        childList(newNode);
    }
    else if (duty == "수강생" && group != 0) { // 수강생일 경우 강의교수를 찾아 부모로 지정하고, 해당 강의교수의 자식 링크드리스트에 추가시킨다.
        for (Child* current = root->headChild; current != NULL; current = current->next) {
            if (current->cNode->info.group == group) {
                if (current->cNode->childNum >= 30) { // 해당 분반의 인원이 30명이 초과되었을 경우 추가시킬 수 없음을 나타낸다.
                    cout << "추가할 수 없음" << endl;
                    delete newNode; // 추가할 수 없으므로 생성한 노드를 다시 삭제한다.
                    break;
                }
                else {
                    newNode->parent = current->cNode;
                    childList(newNode);
                    break;
                }
            }
        }
        else if (current == root->tailChild) { // 찾는 분반의 강의교수가 없을 경우 추가시킬 수 없음을 나타낸다.
            cout << "추가할 수 없음" << endl;
            delete newNode;
            break;
        }
    }
}
```

삽입이 수행될 때 nodeInsert 함수를 통해 인자로 인원의 정보를 받는다. 새로운 노드 newNode를 생성하여 입력 받은 값들을 넣는다. newNode의 직책에 따라 if문을 사용하여 다른 값을 설정해준다.

- newNode의 직책이 책임교수일 경우 : root를 노드로 설정한다.
- newNode의 직책이 강의교수일 경우 : newNode의 부모를 root로 설정해주고 newNode의 주소를 인자로 받는 childList 함수로 들어간다.
- newNode의 직책이 수강생일 경우 : 해당되는 분반을 찾을 때까지 for문을 이용하였다. for문에서 Child형을 가리키는 임시포인트변수 current를 이용해 root의 자식 링크드리스트를 따라 자식의 분반번호를 비교하였다. cuurent가 가리키는 노드와 수강생의 분반번호와 일치할 경우, newNode의 부모를 current의 노드로 설정한다. 그리고 newNode의 주소를 인자로 받는 childList 함수로 들어간다. 하지만 수강생의 인원이 30명이 넘었을 경우에는 추가할 수 없게 하였다. 또한 해당 분반의 강의교수가 존재하지 않을 경우 추가 할 수 없게 하였다.

void Tree::childList(Node* newNode)

```
void Tree::childList(Node* newNode) {
    Child* parChild = new Child; //newNode를 가르키는 부모의 Child

    parChild->cNode = newNode;
    int groupNum = newNode->info.group; //분반번호 = 넘겨받은 자식의 분반번호

    if (newNode->parent->headChild == NULL) { //자식을 처음 생성했을 때
        newNode->parent->headChild = parChild;
        newNode->parent->tailChild = parChild;
        newNode->parent->childNum++;
    }
    else{
        if (newNode->parent == root) {
            for (Child* current = root->headChild; current != NULL; current = current->next) { //
                if (groupNum == current->cNode->info.group) {
                    cout << "추가할 수 없음" << endl;
                    delete newNode;
                    delete parChild;
                    break;
                }
                else if (current == root->tailChild) {
                    parChild->prev = newNode->parent->tailChild;
                    newNode->parent->tailChild->next = parChild;
                    newNode->parent->tailChild = parChild;
                    newNode->parent->childNum++;
                    break;
                }
            }
        }
        else if (newNode->parent != root) {
            parChild->prev = newNode->parent->tailChild;
            newNode->parent->tailChild->next = parChild;
            newNode->parent->tailChild = parChild;
            newNode->parent->childNum++;
        }
    }
}
```

child형 parChild를 생성하고 newNode를 가리키도록 한다. int형 변수 groupNum에 newNode의 분반번호를 저장한다. 만약 newNode가 부모의 첫 자식일 때, 부모의 head Child, tail Child를 parChild로 설정한다. 그리고 자식의 개수를 1 증가시킨다. 첫 자식이 아니고 newNode의 부모가 root일 경우(newNode가 강의교수임을 의미한다) Child형 포인트변수를 이용하여 root의 자식 링크드리스트의 head부터 newNode의 분반번호와 같은 분반이 이미 있는지 확인해 나간다. 생성하려는 번호의 같은 분반이 없을 경우 root의 마지막 자식(tailChild)에 parChild를 추가시킨다. newNode의 부모가 root가 아니라면 newNode는 수강생이므로, newNode의 부모(강의교수)의 tailChild에 parChild를 추가시킨다.

- * 시간 복잡도 : 최악의 경우에 책임교수의 생성은 $O(1)$ 시간에 생성되고, 강의교수, 수강생 생성은 강의교수의 수가 n 일 때 $O(n)$ 시간에 생성된다.
- * 공간 복잡도 : 삽입 연산을 통해 newNode, parChild가 추가될 때 공간의 증가율은 상수이기 때문에 영향을 끼치지 않는다.

2. 삭제

void Tree::Delete(string name)

```

void Tree::Delete(string name) { //이름을 받아 해당 노드를 삭제한다.

    Child* dchild = new Child;
    Node* dnode = new Node;
    delete dnode;
    dchild = NULL;
    dnode = NULL;

    for (Child* i = root->headChild; dnode == NULL; i = i->next) {
        if (name == i->cNode->info.name) { //강의교수 삭제
            if (i->cNode->headChild != NULL) { //삭제하려고 하는 강의
                cout << "삭제할 수 없음" << endl;
                break;
            }
            else if (root->headChild == root->tailChild) { //루트의 자
                root->headChild = NULL;
                root->tailChild = NULL;
            }
            else if (i == root->headChild) { //삭제하는 노드가 루트의
                i->next->prev = root->headChild;
                root->headChild = i->next;
            }
            else if (i == root->tailChild) { //삭제하는 노드가 루트이
                i->prev->next = root->tailChild;
                root->tailChild = i->prev;
            }
            else { //중간 삭제
                i->prev->next = i->next;
                i->next->prev = i->prev;
            }
            root->childNum--;
            dnode = i->cNode;
            dchild = i;
            break;
        }
    }

    for (Child* j = i->cNode->headChild; dnode == NULL; j = j->next) {
        if (j == NULL) break;
        else if (name == j->cNode->info.name) { //수강생 삭제
            i->cNode->childNum--;
            if (i->cNode->headChild == i->cNode->tailChild) {
                i->cNode->headChild = NULL;
                i->cNode->tailChild = NULL;
            }
            else if (j == i->cNode->headChild) {
                j->next->prev = i->cNode->headChild;
                i->cNode->headChild = j->next;
            }
            else if (j == i->cNode->tailChild) {
                j->prev->next = i->cNode->tailChild;
                i->cNode->tailChild = j->prev;
            }
            else {
                j->prev->next = j->next;
                j->next->prev = j->prev;
            }
            dnode = j->cNode;
            dchild = j;
            break;
        }
        else if (j == i->cNode->tailChild) {
            break;
        }
    }
    if (dnode != NULL) break;
    if (i == root->tailChild) {
        cout << "삭제할 수 없음" << endl; //삭제하려고 하는 강의교수
        break;
    }
}
delete dnode; //삭제할 강의의 노드를 삭제한다.
delete dchild;
}

```

사용자로부터 이름을 받아온다. 삭제할 노드와 그 노드를 자식으로 가리키는 Child를 삭제하기 위해 새로운 dchild와 dnode를 만들고 NULL값으로 초기화 시킨다. 이중 for문을 이용하여 root의 첫번째 자식부터 preorder traversal 형태로 각 노드를 방문한다. 같은 이름의 노드가 존재할 때 그 부모의 자식 링크드리스트에서 연결을 끊는다. 이 때 이 child이 링크드리스트에서 head, tail 부분에 존재하는지 구분하여 지정해주었다. 링크드리스트 내 head에 존재하면 next child의 prev 값은 head를 지정하고, head를 next child으로 지정하게 하였다. Tail에 존재하면 prev child의 next를 tail로 설정하고 tail을 prev child로 설정하게 했다. 중간의 삭제는 child의 prev child와 next child가 서로를 가리키도록 만들어준다. Child가 링크드리스트에서 제외된 후 링크드리스트의 자식 개수(num)을 1 감소시키고 dnode와 dchild에 각 노드와 child를 넣는다. dnode와 dchild에 삭제할 노드와 자식을 넣고 이중 for문을 종료한다. 만약 찾으려는 이름의 노드가 root의 tail까지 없을 경우 "삭제할 수 없음"을 출력한다. 반복문을 빠져나오면 delete를 이용하여 지정된 dnode와 dchild에 해당하는 주소값의 객체를 삭제한다.

- * 시간 복잡도 : 최악의 경우 찾고자 하는 노드가 root의 tailChild에 해당되는 노드의 tailChild 노드이거나 해당되는 이름을 찾지 못하고 반복문이 끝날 때이다. 노드의 개수 N만큼 모든 노드를 탐색하므로 최악의 수행시간은 $O(N)$ 이다.
- * 공간 복잡도 : 노드가 삭제될 경우 메모리를 추가로 늘리지 않으므로 영향을 끼치지 않는다.

3. 출력

void Tree::groupPrint(int group)

```
void Tree::groupPrint(int group) {
    if (group == 0) { //전체출력
        cout << root->info.name << " " << root->info.group << " " <<
            root->info.duty << " " << root->info.id << endl; //책임교수 출력
        for (Child* current = root->headChild; current != NULL; current = current->next) {
            Print(current->cNode, current->cNode->headChild); //자식노드를 출력하는 함수
            if (current == root->tailChild)
                break;
        }
    }
    else //분반출력
        for (Child* current = root->headChild; current != NULL; current = current->next) {
            if (current->cNode->info.group == group) {
                Print(current->cNode, current->cNode->headChild);
                break;
            }
            else if (current == root->tailChild) { //찾는 분반이 없을 때 나타낸다.
                cout << "없음" << endl;
                break;
            }
        }
}
```

사용자로의 분반번호를 받아온다.

- **분반번호가 0일 경우** : root의 정보(이름, 분반번호, 직책, id)를 출력시키고, for문을 이용하여 root의 자식 링크드리스트 head부터 시작하여 tail을 가르킬 때까지 child를 next한다. 이 때 링크드리스트에 포함된 각 노드와 그 노드의 head child를 Print 함수의 인자로 넣는다.
- **분반번호가 0이 아닌 이외의 경우** : 인자로 넘어 온 분반번호를 for문을 이용해 root의 자식링크드리스트에서 찾는다. 노드의 분반번호가 root의 자식의 분반번호와 일치할 경우 해당 분반의 노드(강의교수)와 노드의 headChild를 print 함수에 인자로 넣는다. 만약 Root의 tailChild까지 분반번호를 찾지 못하였다면 찾는 분반이 없음을 나타내는 "없음"을 출력한다.

void Tree::Print(Node* cNode, Child* printChild)

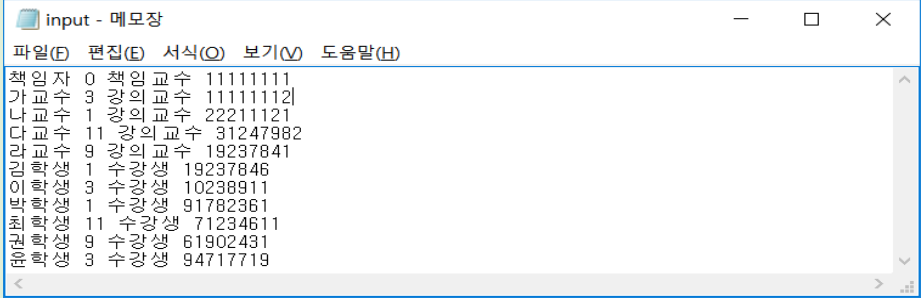
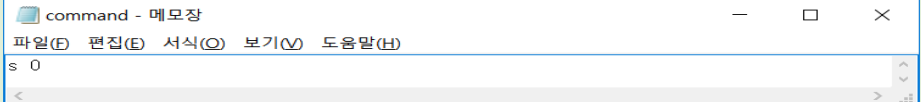
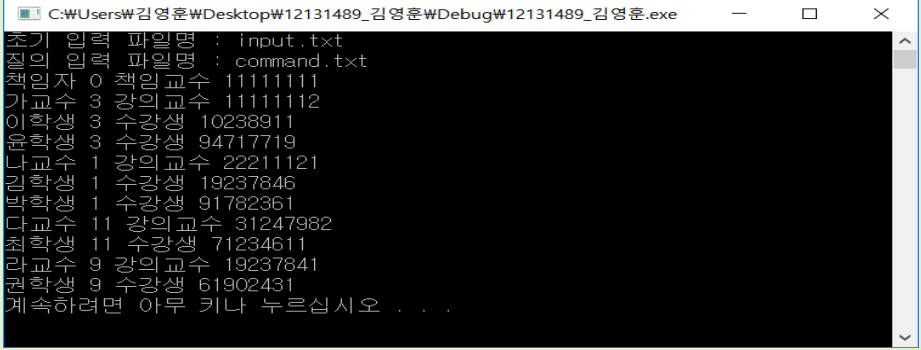
```
void Tree::Print(Node* cNode, Child* printChild) { //preorder traversal
    cout << cNode->info.name << " " << cNode->info.group << " " <<
        cNode->info.duty << " " << cNode->info.id << endl;
    if (printChild == NULL) {} //자식이 없을경우 중지
    else{
        cNode = printChild->cNode;
        printChild = printChild->next;
        Print(cNode, printChild);
    }
}
```

인자로 받은 노드의 정보를 출력하고 노드의 headChild가 없을 경우 함수가 종료되고, 그렇지 않을 경우 노드에 child가 가리키는 노드의 주소를 넣고, child는 연결된 다음 child를 가리킨다
그리고 다시 Print 함수에 노드와 child를 인자로 넣어 불러온다. (재귀)

- * 시간복잡도 : 0을 입력받았을 경우 노드의 수 N만큼 반복문과 재귀함수를 통해 출력하므로 최악의 경우 $O(N)$ 이다. 0이 아닐 때 해당 분반을 찾기 위해 최악의 경우 강의교수의 수가 A일 때 $O(A)$ 만큼 걸리고, Print 함수는 자식의 개수 B일 때 자식의 개수만큼 출력하므로 $O(B)$ 시간이 걸린다.
- * 공간복잡도 : 출력기능만 가지고 있기 때문에 노드와 자식의 값을 변경하지 않아 $O(0)$ 에 수행된다.

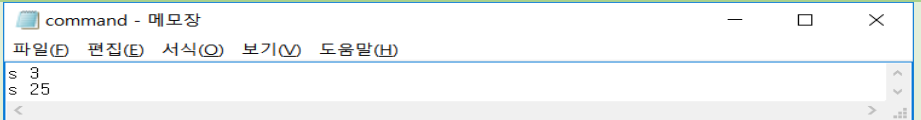
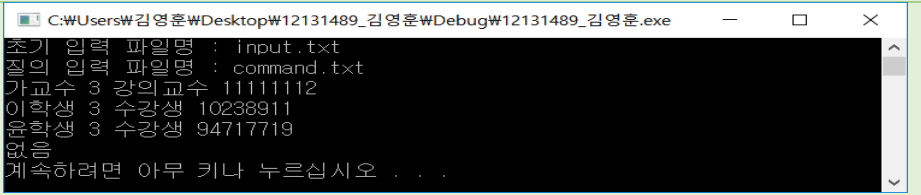
4. 인터페이스 및 사용법

Input.txt 초기값 입력, command.txt 전체 노드를 확인하는 인터페이스 입력

input.txt	
command.txt	
출력 화면	

Input.txt에 초기값을 넣고 command.txt에 전체 노드를 탐색하는 s 0 입력을 입력하였다.
출력으로 초기값의 모든 노드들이 나왔다.

s 분반 확인

command.txt	
출력 화면	

인터페이스 s(탐색)를 통해 3분반의 목록을 출력하였다.
25분반에 소속된 인원은 없으므로 "없음"이 출력되었다.

i 구성원 추가

command.txt

```
command - ...
파일(F) 편집(E) 서식(O) 보기(V)
도움말(H)
i 학생1 25 수강생 19201001
i 마교수 25 강의교수 10101010
i 학생1 25 수강생 19201001
i 학생2 25 수강생 19201002
i 학생3 25 수강생 19201003
i 학생4 25 수강생 19201004
i 학생5 25 수강생 19201005
i 학생6 25 수강생 19201006
i 학생7 25 수강생 19201007
i 학생8 25 수강생 19201008
i 학생9 25 수강생 19201009
i 학생10 25 수강생 19201010
i 학생11 25 수강생 19201011
i 학생12 25 수강생 19201012
i 학생13 25 수강생 19201013
i 학생14 25 수강생 19201014
i 학생15 25 수강생 19201015
i 학생16 25 수강생 19201016
i 학생17 25 수강생 19201017
i 학생18 25 수강생 19201018
i 학생19 25 수강생 19201019
i 학생20 25 수강생 19201020
i 학생21 25 수강생 19201021
i 학생22 25 수강생 19201022
i 학생23 25 수강생 19201023
i 학생24 25 수강생 19201024
i 학생25 25 수강생 19201025
i 학생26 25 수강생 19201026
i 학생27 25 수강생 19201027
i 학생28 25 수강생 19201028
i 학생29 25 수강생 19201029
i 학생30 25 수강생 19201030
i 학생31 25 수강생 19201031
s 25
```

출력 화면

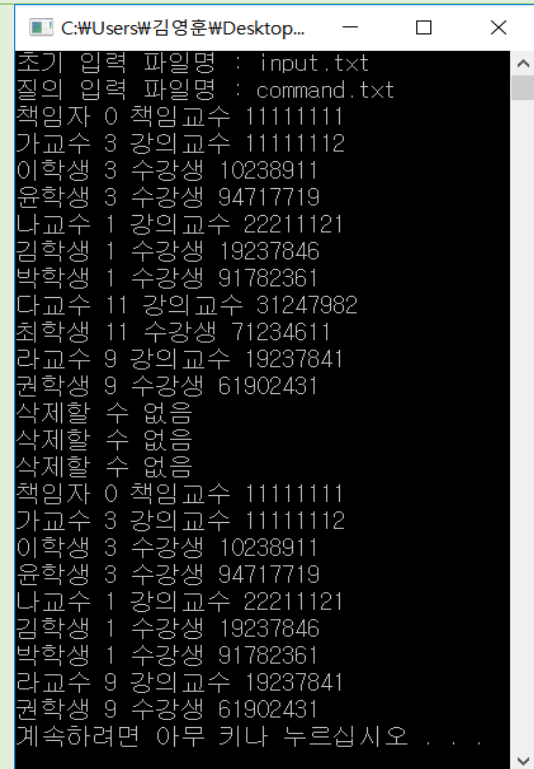
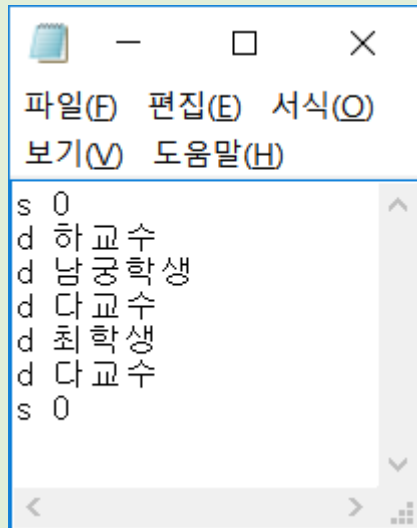
```
C:\Users\김영훈\Desktop\...
초기 입력 파일명 : input.txt
질의 입력 파일명 : command.txt
추가할 수 없음
추가할 수 없음
마교수 25 강의교수 10101010
학생1 25 수강생 19201001
학생2 25 수강생 19201002
학생3 25 수강생 19201003
학생4 25 수강생 19201004
학생5 25 수강생 19201005
학생6 25 수강생 19201006
학생7 25 수강생 19201007
학생8 25 수강생 19201008
학생9 25 수강생 19201009
학생10 25 수강생 19201010
학생11 25 수강생 19201011
학생12 25 수강생 19201012
학생13 25 수강생 19201013
학생14 25 수강생 19201014
학생15 25 수강생 19201015
학생16 25 수강생 19201016
학생17 25 수강생 19201017
학생18 25 수강생 19201018
학생19 25 수강생 19201019
학생20 25 수강생 19201020
학생21 25 수강생 19201021
학생22 25 수강생 19201022
학생23 25 수강생 19201023
학생24 25 수강생 19201024
학생25 25 수강생 19201025
학생26 25 수강생 19201026
학생27 25 수강생 19201027
학생28 25 수강생 19201028
학생29 25 수강생 19201029
학생30 25 수강생 19201030
계속하려면 아무 키나 누르십시오 . . .
```

인터페이스 i(삽입)를 통해 25분반에 학생1 수강생을 추가하려고 했지만 해당 강의교수가 없어 "추가할 수 없음"이 출력되었다. 이후 강의교수가 추가되고 30번째 학생까지 추가되었다. 31번째 학생은 학생수가 30명을 넘어 "추가할 수 없음"이 출력되었다. s 25를 통해 25분반의 인원이 강의교수부터 30번째 학생까지 제대로 입력되었음을 출력하였다.

d 구성원 삭제

command.txt

출력 화면



인터페이스 d "이름" 을 입력하면 해당되는 이름의 노드를 찾아가 삭제한다. "하교수", "남궁학생"이라는 이름의 노드는 존재하지 않아 "삭제할 수 없음"이 출력되었다. "다교수"는 강의교수로 수강생들이 있어 삭제할 수 없음이 출력되었다. "최학생"이 삭제되고 11분반의 수강생이 존재하지 않아 "다교수"를 삭제할 수 있고 인터페이스를 통해 삭제되었다. 삭제 전후의 전체 노드(s 0)을 통해 삭제 되었음을 비교할 수 있다..

5. 평가 및 개선 방향

1. 장점

링크드리스트를 이용하여 트리를 구현하여 낭비되는 공간이 없어 배열보다 공간상 이득을 볼 수 있다.

2. 단점

코드를 작성할 때 대부분 if문과 for문을 통해 만들어 간결성과 가독성이 떨어진다. 해당 분반을 찾을 때 노드가 분반번호로 정렬이 되어있지 않아 매 번 headChild부터 해당 노드의 분반번호와 비교를 하여야 한다.

3. 향후 개선 방향

프로그램에서 root의 child(강의교수)들을 삽입할 때 분반순으로 정렬하면, 분반을 찾아 출력하거나 수강생들을 삽입할 때 더 빠를 수 있다고 생각한다. 그리고 중복되는 코드를 함수로 구현하여 중복을 피하고 간결성을 늘려야 한다.