**9주차 결과보고서**

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**1) createRankList():** 이 함수는 Input 파일인 "rank.txt" 파일에서 랭킹 정보를 읽어와서 LinkedList로 저장하는 과정을 수행한다. 파일을 순차적으로 읽으며, 새로운 노드를 생성하여 연결 리스트에 추가한다. 총 시간 복잡도와 공간 복잡도는 O(n)이며 여기서 n은 랭킹 정보의 수이다. 알고리즘은 다음과 같다.

1. "rank.txt" 파일 열기.
2. 파일에서 랭킹 정보의 수를 읽어와서 head 구조체의 length 변수에 저장한다.
3. 첫 번째 랭킹 정보를 읽어와서 새로운 노드(first)를 생성하고, 읽어온 정보를 노드에 저장한다. 노드의 link를 NULL로 설정한다. head의 first를 새로운 노드로 설정한다.
4. 반복문을 통해 head의 length - 1만큼 다음 랭킹 정보를 읽어온다.

새로운 노드(rank)를 생성하고, 읽어온 정보를 노드에 저장한다.

현재 노드(curr)를 head의 first로 초기화한다.

반복문을 통해 curr를 i번째까지 이동시킨다.

curr의 link를 새로운 노드(rank)로 설정한다.

1. score\_number 변수를 head의 length로 설정한다.
2. 파일을 닫는다.

*(구현 코드)*

void createRankList() {

    FILE \*fp;

    int i, j;

    char str[NAMELEN + 1];

    int num;

    int read;

    int index = 0;

    score\_number = 0;

    // 1. "rank.txt"열기

    fp = fopen("rank.txt", "r");

    // 2. 파일에서 랭킹 정보 읽어오기

    // 3. LinkedList로 저장

    fscanf(fp, "%d", &head->length); // 랭킹 수 읽어오기

    fscanf(fp, "%s %d", str, &num); // 첫 번째 랭킹 정보 읽어오기

    Node\* first;

    first = malloc(sizeof(Node));

    strcpy(first->name, str);

    first->score = num;

    first->link = NULL;

    head->first = first;

    score\_number++;

    for (int i = 0; i < head->length - 1; i++) {

        // 순서대로 랭킹 정보 읽어오기

        read = fscanf(fp, "%s %d", str, &num);

        // 새로운 노드 생성

        Node\* rank;

        rank = malloc(sizeof(Node));

        strcpy(rank->name, str);

        rank->score = num;

        rank->link = NULL;

        // 새로운 노드 리스트에 연결

        Node\* curr = head->first;

        for (int j = 0; j < i; j++) {

            curr = curr->link;

        }

        curr->link = rank;

        score\_number++;

    }

    score\_number = head->length;

    // 4. 파일 닫기

    fclose(fp);

}

**2) rank():** 이 알고리즘은 사용자가 rank 메뉴에서 선택한 작업에 따라서 출력과 랭킹 삭제 등 다양한 동작을 수행합니다. 각 메뉴에 대한 동작은 별도의 조건문으로 구분되어 처리되며, 시간 복잡도는 O(n), 공간 복잡도는 O(1)이다. rank 함수의 전체 알고리즘은 다음과 같다.

1. X와 Y 변수를 1로 초기화하고, ch, i, j 변수를 선언한다.
2. 화면을 지우고(printw("clear")), 3개의 메뉴를 출력한다.
3. wgetch() 함수를 사용하여 사용자로부터 메뉴 번호를 입력받아 변수 ch에 저장한다.
4. 각 메뉴에 따라 입력받을 값을 변수에 저장한다.

메뉴 1: X와 Y를 입력받고, 적절한 입력인지 확인한 후(X <= Y), X부터 Y까지의

랭킹을 출력한다.

메뉴 2: 문자열을 입력받아 저장된 이름과 비교하여 해당하는 랭킹을 출력한다.

메뉴 3: 랭크 번호를 입력받아 리스트에서 해당 랭킹을 삭제한다.

1. 해당 랭킹이 존재하는지 확인하고, 존재할 경우 노드를 찾아 삭제한다.

첫 번째 랭킹을 삭제하는 경우와 두 번째 이후의 랭킹을 삭제하는 경우를

구분하여 처리한다.

삭제 후, score\_number 변수를 업데이트하고, writeRankFile() 함수를 호출하여

"rank.txt" 파일을 업데이트한다.

1. getch() 함수를 사용하여 사용자의 임의의 키 입력을 대기한다.

rank 함수의 메뉴별 알고리즘은 다음과 같다.

* 메뉴 1 (ch == '1'):

변수 X와 Y를 입력받는다.

랭킹 목록을 출력하기 위해 화면을 초기화하고 헤더를 출력한다.

입력된 X와 Y 사이의 랭킹을 출력한다. 랭킹은 노드의 인덱스와 일치한다.

입력된 X와 Y가 유효하지 않은 범위인 경우 오류 메시지를 출력한다.

* 메뉴 2 (ch == '2'):

사용자로부터 이름을 입력받는다.

랭킹 목록을 출력하기 위해 화면을 초기화하고 헤더를 출력한다.

입력된 이름과 동일한 이름을 가진 랭킹을 출력한다.

일치하는 사용자가 없는 경우 에러 메시지를 출력한다.

* 메뉴 3 (ch == '3'):

사용자로부터 삭제할 랭크 번호를 입력받는다.

입력된 랭크 번호의 유효성을 검사한다.

유효한 랭크 번호인 경우, 해당 랭크의 노드를 삭제하고 랭킹 수를 업데이트한다.

삭제 후, "rank.txt" 파일을 업데이트한다.

*(구현 코드)*

void rank() {

    //목적: rank 메뉴를 출력하고 점수 순으로 X부터~Y까지 출력함

    //1. 문자열 초기화

    int X = 1, Y=score\_number, ch, i, j;

    clear();

    //2. printw()로 3개의 메뉴출력

    printw("1. list ranks from X to Y\n");

    printw("2. list ranks by a specific name\n");

    printw("3. delete a specific rank\n");

    //3. wgetch()를 사용하여 변수 ch에 입력받은 메뉴번호 저장

    ch = wgetch(stdscr);

    //4. 각 메뉴에 따라 입력받을 값을 변수에 저장

    //4-1. 메뉴1: X, Y를 입력받고 적절한 input인지 확인 후(X<=Y), X와 Y사이의 rank 출력

    if (ch == '1') {

        int scan = 0;

        // X 입력받기

        printw("X: ");

        echo();

        scan = scanw("%d", &X);

        if (scan == -1) X = 1;

        noecho();

        // Y 입력받기

        printw("Y: ");

        echo();

        scan = scanw("%d", &Y);

        if (scan == -1) Y = score\_number;

        noecho();

        // 랭크 목록 시작

        printw("       name       |   score   \n");

        printw("------------------------------\n");

        // 적절한 input인지 확인

        if (X > Y) {

            printf("search failure: no rank in the list\n");

        }

        // X와 Y사이의 rank 출력

        else {

            int index = 0;

            Node\* curr;

            curr = head->first;

            int space;

            while(curr) {

                if ((index + 1 >= X)&&(index + 1 <= Y)) {

                    printw(" %s", curr->name);

                    space = 17 - strlen(curr->name);

                    for (int i = 0; i < space; i++) {

                        printw(" ");

                    }

                    printw("|");

                    printw(" %d\n", curr->score);

                }

                index++;

                curr = curr->link;

            }

        }

    }

    //4-2. 메뉴2: 문자열을 받아 저장된 이름과 비교하고 이름에 해당하는 리스트를 출력

    else if ( ch == '2') {

        char str[NAMELEN+1];

        int check = 0;

        // 사용자의 이름(문자열) 입력받기

        printw("input the name: ");

        echo();

        scanw("%s", str);

        noecho();

        // 랭크 목록 시작

        printw("       name       |   score   \n");

        printw("------------------------------\n");

        // 저장된 이름과 비교하기

        Node\* curr;

        curr = head->first;

        int space;

        while(curr) {

            if (strcmp(str, curr->name) == 0) {

                printw(" %s", curr->name);

                space = 17 - strlen(curr->name);

                for (int i = 0; i < space; i++) {

                    printw(" ");

                }

                printw("|");

                printw(" %d\n", curr->score);

                check = 1;

            }

            curr = curr->link;

        }

        // 일치하는 사용자가 없으면 에러 출력

        if (check == 0) {

            printw("\nsearch failure: no name in the list\n");

        }

    }

    //4-3. 메뉴3: rank번호를 입력받아 리스트에서 삭제

    else if ( ch == '3') {

        int X;

        // rank 번호 입력받기

        printw("input the rank: ");

        echo();

        scanw("%d", &X);

        noecho();

        // 입력받은 랭킹이 존재하는지 확인

        if ((X > score\_number) || (X < 0)) {

            printw("\nsearch failure: the rank not in the list\n");

        }

        // 해당 랭킹이 범위를 벗어나지 않으면 노드를 찾아 삭제

        else {

            int count = 0;

            Node\* curr = head->first;

            Node\* pre = curr;

            // 첫 번째 랭킹을 삭제하는 경우

            if (X == 1) {

                head->first = curr->link;

                free(curr);

                printw("\nresult: the rank deleted\n");

                score\_number--;

            }

            // 두 번째 ~ 마지막 랭킹을 삭제하는 경우

            else {

                while (curr) {

                    count++;

                    if(count == X) {

                        pre->link = curr->link;

                        free(curr);

                        printw("\nresult: the rank deleted\n");

                        score\_number--;

                        break;

                    }

                    pre = curr;

                    curr = curr->link;

                }

            }

            head->length = score\_number;

        }

        writeRankFile();

    }

    getch();

}

**3) writeRankFile():** 이 함수는 추가된 랭킹 정보가 있으면 해당 정보를 "rank.txt" 파일에 쓴다. 이때 랭킹 정보의 수와 각 랭킹 정보를 파일에 기록하는 작업을 수행한다. 시간 복잡도는 O(n)이고 공간 복잡도는 O(1)이다. 알고리즘은 다음과 같다.

1. "rank.txt" 파일을 쓰기 모드로 연다.
2. 랭킹 정보의 수를 "rank.txt" 파일에 기록한다.
3. 랭킹 정보를 가리키는 현재 노드 변수 curr을 헤드의 첫 번째 노드로 초기화한다.
4. curr이 NULL이 아닐 때까지 반복한다.

현재 노드의 이름과 점수를 "rank.txt" 파일에 기록한다.

curr을 다음 노드로 이동한다.

1. "rank.txt" 파일을 닫는다.

*(구현 코드)*

void writeRankFile() {

    // 목적: 추가된 랭킹 정보가 있으면 새로운 정보를 "rank.txt"에 쓰고 없으면 종료

    Node\* curr;

    //1. "rank.txt" 연다

    FILE \*fp = fopen("rank.txt", "w");

    //2. 랭킹 정보들의 수를 "rank.txt"에 기록

    fprintf(fp, "%d\n", score\_number);

    //3. 탐색할 노드가 더 있는지 체크하고 있으면 다음 노드로 이동, 없으면 종료

    curr = head->first;

    while(curr) {

        fprintf(fp, "%s %d\n", curr->name, curr->score);

        curr = curr->link;

    }

    // 파일 닫기

    fclose(fp);

}

**4) newRank():** 이 함수는 GameOver 시 호출되어 사용자 이름을 입력받고 해당 점수와 함께 리스트의 적절한 위치에 저장하는 기능을 수행한다. 시간 복잡도는 O(n)이고, 공간 복잡도는 O(1)이다. 알고리즘은 다음과 같다.

1. 사용자에게 이름을 입력받는다.
2. 새로운 노드를 생성하여 이름과 점수를 저장한다.
3. 리스트의 적절한 위치에 노드를 저장한다.

만약 리스트가 비어있다면, 새로운 노드를 헤드로 설정하고 리스트의 길이를

증가시킨다.

그렇지 않은 경우, 현재 노드와 이전 노드 변수를 초기화한다.

노드의 점수를 비교하여 위치를 결정하고, 해당 위치에 새로운 노드를 삽입한다.

리스트의 길이를 증가시킨다.

1. writeRankFile 함수를 호출하여 업데이트된 랭킹 정보를 파일에 저장한다.

*(구현 코드)*

void newRank(int score) {

    // 목적: GameOver시 호출되어 사용자 이름을 입력받고 score와 함께 리스트의 적절한 위치에 저장

    char str[NAMELEN+1];

    clear();

    //1. 사용자 이름을 입력받음

    printw("your name: ");

    echo();

    scanw("%s", str);

    noecho();

    //2. 새로운 노드를 생성해 이름과 점수를 저장

    Node\* new;

    new = malloc(sizeof(Node));

    strcpy(new->name, str);

    new->score = score;

    new->link = NULL;

    //3. 리스트의 적절한 위치에 저장

    Node\* curr = head->first;

    Node\* pre = head->first;

    if (score\_number == 0) {

        head->first = new;

        score\_number++;

    }

    else {

        int index = 0;

        // 점수 비교해서 index 얻기

        while (curr) {

            if ((new->score) < (curr->score)) {

                curr = curr->link;

                index++;

            }

            else {

                break;

            }

        }

        // 적절한 위치에 삽입

        for (int j = 0; j < index - 1; j++) {

            pre = pre->link;

        }

        new->link = pre->link;

        pre->link = new;

        score\_number++;

    }

    head->length = score\_number;

    writeRankFile();

}

본인이 선택한 랭킹 시스템을 구현하기 위한 자료구조가 왜 효율적인지 시간 및 공간 복잡도를 통해 보이고, 설명하시오. -> 3-1)에서 설명

**2.**

전반적으로 Linked List라는 자료구조에 대한 이해도가 올라갔다. 특히 각 노드의 link를 다루는 법을 습득했고 segmentation fault 없이 노드를 생성하고 연결하고 삭제하는 방법에 대해 어느 정도 이해한 것 같다. 노드를 삽입, 삭제할 때 노드의 수에 따라 경우를 나눠서 진행해야된다는 점도 알게되었다. 또한 리스트의 노드들을 값에 따라 순차적으로 정렬하려면 while문과 if문을 어떻게 다뤄야 하는지에 대해서도 배웠다. 마지막으로 printw(), scanw(), echo(), noecho(), getch() 등 ncurse 라이브러리의 다양한 함수들의 사용법에 대해 알게된 점도 좋았다.

**3.**

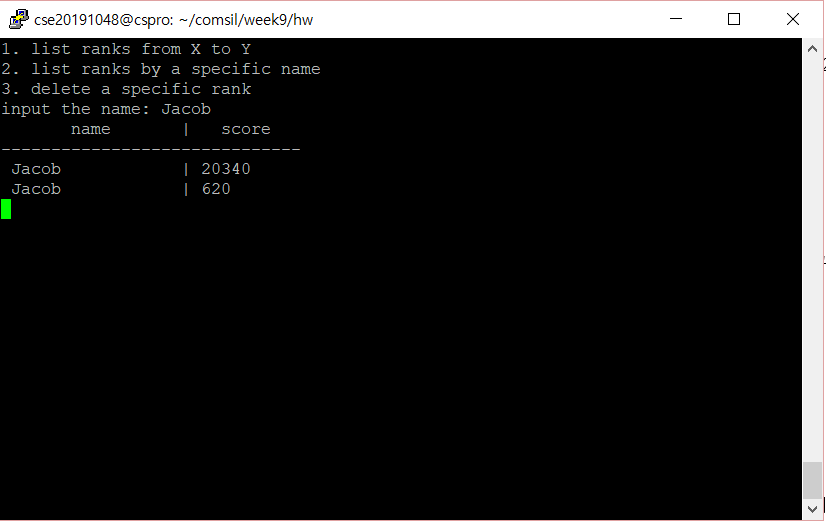
3-1) 선택한 자료구조의 효율성을 평가하고 왜 그렇게 생각하는지 상세하게 기술한다.

랭킹 시스템의 주요 기능에 대한 Linked List의 시간 및 공간 복잡도는 다음과 같다.

1. Game over가 되고, 새로운 랭킹 정보(사용자 이름, 점수)가 등록될 때, 시간 복잡도는 O(n)이고 공간 복잡도는 O(1)이다.
2. 원하는 랭킹 범위를 입력 받고, 랭킹을 추출하기 위한 과정에서 자료 구조 탐색 및 랭킹 추출에서의 시간 복잡도는 O(n)이고, 공간 복잡도는 O(1)이다.

Linked List의 이러한 시간, 공간 복잡도를 다른 자료구조와 비교해보자. 먼저 배열은 인덱스를 사용하여 원소에 접근하는 것이 빠르고 메모리를 연속적으로 사용하기 때문에 접근 시간이 상수라는 점에서 Linked List보다 강점을 가진다. 그러나 삽입과 삭제 연산은 배열의 크기를 변경하고 데이터를 이동해야 하므로 더 많은 시간이 소요될 수 있다. 또한 정렬 연산도 Linked List에 비해 더 많은 시간이 든다. 다음으로 이진 검색 트리는 노드가 정렬된 순서로 연결되어 있으며, 탐색, 삽입, 삭제 연산이 평균적으로 O(log n)의 시간 복잡도를 가진다. 이 자료구조는 특히 정렬된 데이터를 효율적으로 관리하는데 강점을 가지기에 Linked List보다 탐색, 삽입, 삭제 측면에서 효율적이라고 볼 수 있다. 정리하자면 다른 자료구조들과 비교했을 때 Linked List는 배열보다는 효율적이고 이진 탐색 트리보다는 비효율적인 자료구조라고 볼 수 있다.

3-2) 사용자의 이름을 입력하고 해당하는 랭킹 정보 출력 화면 첨부



3-3) 사용자 이름으로 원하는 사용자의 이름을 검색할 때의 시간 복잡도는 O(n), 공간 복잡도는 O(1)이다. 먼저 시간 복잡도의 경우 문자열을 입력받고 저장된 이름과 비교하는 부분은 노드의 수에 비례하여 선형으로 실행된다. 따라서, 노드의 수를 n이라고 할 때, 비교하는 부분의 시간 복잡도는 O(n)이다. 다음으로 공간 복잡도의 경우 코드에서 사용되는 공간은 고정된 크기의 배열(str)과 변수(check, curr, space)들뿐이다. 입력받는 문자열의 크기에 관계없이 공간의 크기는 일정하므로, 공간 복잡도는 O(1)이다. 코드는 다음과 같다.

*(구현 코드)*

else if ( ch == '2') {

char str[NAMELEN+1];

int check = 0;

// 사용자의 이름(문자열) 입력받기

printw("input the name: ");

echo();

scanw("%s", str);

noecho();

// 랭크 목록 시작

printw(" name | score \n");

printw("------------------------------\n");

// 저장된 이름과 비교하기

Node\* curr;

curr = head->first;

int space;

while(curr) {

if (strcmp(str, curr->name) == 0) {

printw(" %s", curr->name);

space = 17 - strlen(curr->name);

for (int i = 0; i < space; i++) {

printw(" ");

}

printw("|");

printw(" %d\n", curr->score);

check = 1;

}

curr = curr->link;

}

// 일치하는 사용자가 없으면 에러 출력

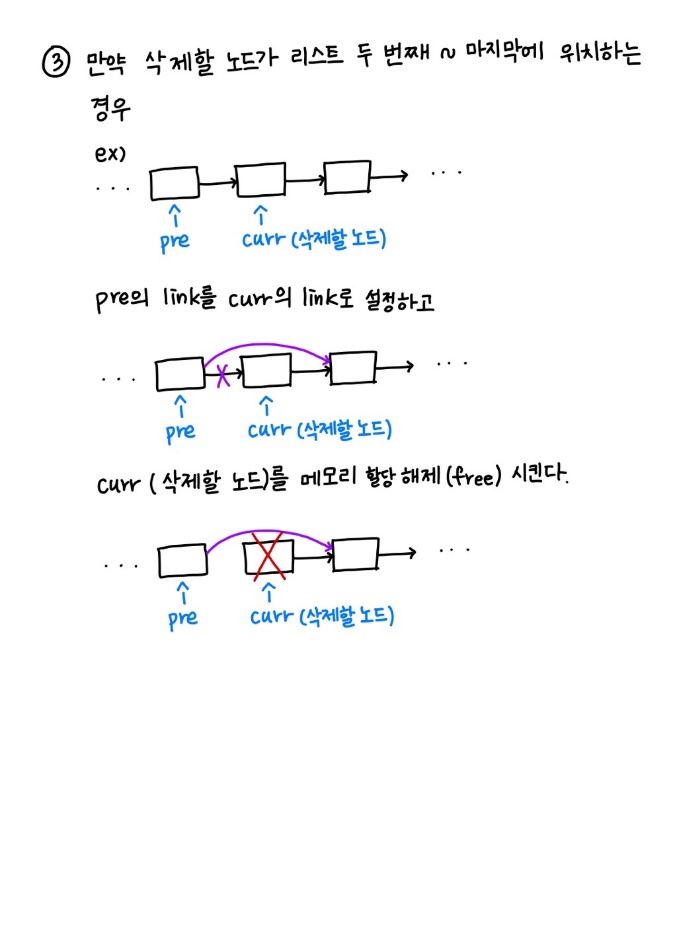
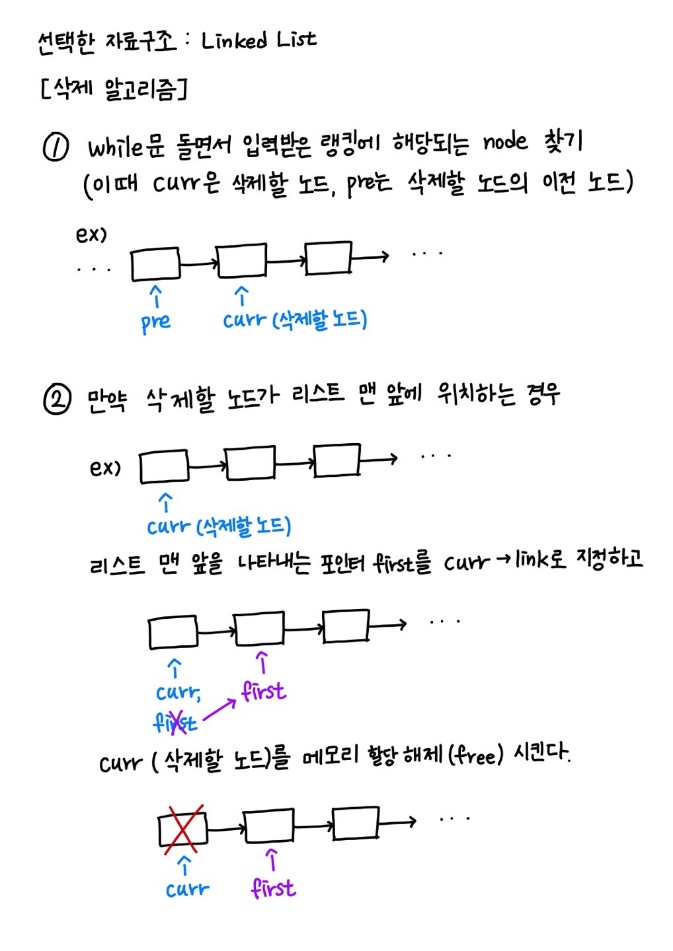
if (check == 0) {

printw("\nsearch failure: no name in the list\n");

}

}

3-4) 선택한 자료구조에 맞춰 삭제 알고리즘을 그림으로 표현한다.



*(구현 코드)*

else if ( ch == '3') {

int X;

// rank 번호 입력받기

printw("input the rank: ");

echo();

scanw("%d", &X);

noecho();

// 입력받은 랭킹이 존재하는지 확인

if ((X > score\_number) || (X < 0)) {

printw("\nsearch failure: the rank not in the list\n");

}

// 해당 랭킹이 범위를 벗어나지 않으면 노드를 찾아 삭제

else {

int count = 0;

Node\* curr = head->first;

Node\* pre = curr;

// 첫 번째 랭킹을 삭제하는 경우

if (X == 1) {

head->first = curr->link;

free(curr);

printw("\nresult: the rank deleted\n");

score\_number--;

}

// 두 번째 ~ 마지막 랭킹을 삭제하는 경우

else {

while (curr) {

count++;

if(count == X) {

pre->link = curr->link;

free(curr);

printw("\nresult: the rank deleted\n");

score\_number--;

break;

}

pre = curr;

curr = curr->link;

}

}

head->length = score\_number;

}

writeRankFile();

}