9주차 결과보고서

전공: 경제학과 학년: 4학년 학번: 20180501 이름: 김연수

**자료구조와 알고리즘 설명:**

자료구조는 LinkedList를 활용했다. 각 노드의 틀은 아래와 같다. 각 노드들은 name, score, nextnode를 가리키는 포인터를 field로 가진다.

typedef struct \_RankNode{

char name[NAMELEN];

int score;

struct \_RankNode \*NextNode;

} RankNode;

각 기능에 대한 알고리즘:

1. createRankList
2. void createRankList() {
3. // 목적: Input파일인 "rank.txt"에서 랭킹 정보를 읽어들임, 읽어들인 정보로 랭킹 목록 생성
4. // Open the rank.txt file
5. FILE \*testfp = fopen("rank.txt", "a");
6. fclose(testfp);
7. FILE \*fp = fopen("rank.txt", "r");
8. // Read the number of ranks
9. fscanf(fp, "%d", &rankCount);
11. // Read and store ranking information
12. // EOF(End Of File): 실제로 이 값은 -1을 나타냄, EOF가 나타날때까지 입력받아오는 if문
13. for (int i = 0; i < rankCount; i++) {
14. RankNode \*newNode = (RankNode \*)malloc(sizeof(RankNode));
15. fscanf(fp, "%s %d", newNode->name, &newNode->score);
16. newNode->NextNode = NULL;
17. if (head == NULL) {
18. head = newNode;
19. } else {
20. RankNode \*curr = head;
21. while (curr->NextNode != NULL) {
22. curr = curr->NextNode;
23. }
24. curr->NextNode = newNode;
25. }
26. }
27. // Close the file
28. fclose(fp);
29. }

createRankList 함수는 rank.txt 파일로부터 rank정보를 입력 받아와서 LinkedList의 각 노드에 rank 정보를 저장해주는 함수다. play를 시작하기 전에 rank정보를 받아오도록 해주기 위해서 main함수에서 play전에 미리 호출해준다.

createRankList함수 구조는 다음과 같다. 먼저 rank.txt파일을 오픈한다. 오픈한 파일에서 첫 째 줄에 있는 rankCount를 읽는다. rankCount 수 만큼 loop를 돌면서, newnode를 생성한 후 그 안에 rank정보를 담아 넣어준다. 그리고 맨 마지막 nextnode 값으로 NULL을 가지고 있는 노드 바로 뒤에 연결한다. 여기서 주의할 점은 rank.txt파일이 없는 경우 rank.txt파일을 생성하도록 해야한다. 때문에 아래와 같이 코드를 작성했다.

// Open the rank.txt file

FILE \*testfp = fopen("rank.txt", "a");

fclose(testfp);

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

2)rank

1. void rank(){
2. //목적: rank 메뉴를 출력하고 점수 순으로 X부터~Y까지 출력함
3. //1. 문자열 초기화
4. int X=1, Y=rankCount, ch, i, j;
5. clear();
6. //2. printw()로 3개의 메뉴출력
7. printw("1. ranks from X to Y\n");
8. printw("2. by a sprecific name\n");
9. printw("3. delete a specific rank\n");
10. //3. wgetch()를 사용하여 변수 ch에 입력받은 메뉴번호 저장
11. ch = wgetch(stdscr);
12. //4. 각 메뉴에 따라 입력받을 값을 변수에 저장
13. //4-1. 메뉴1: X, Y를 입력받고 적절한 input인지 확인 후(X<=Y), X와 Y사이의 rank 출력
14. if (ch == '1') {
15. int x = 1, y = 10000;
16. RankNode \*temp;
17. echo();
18. printw("X: ");
19. scanw("%d", &x);
20. printw("Y: ");
21. scanw("%d", &y);
22. noecho();
23. printw(" name | score \n");
24. printw("----------------------------------\n");
25. temp = head;
26. if(x <= y) {
27. for(i = X; i <= Y; i++) {
28. if(i >= x && i <= y) {
29. printw("%-20s | %d\n", temp->name, temp->score);
30. }
31. if(i > y) break;
32. temp = temp->NextNode;
33. }
34. } else {
35. printw("search failure: no rank in the list\n");
36. }
37. }
38. //4-2. 메뉴2: 문자열을 받아 저장된 이름과 비교하고 이름에 해당하는 리스트를 출력
39. else if ( ch == '2') {
40. char str[NAMELEN+1];
41. int check = 0;
42. RankNode \*temp;
43. printw("input the name: ");
44. echo();
45. scanw("%s", str);
46. noecho();
47. printw(" name | score \n");
48. printw("-----------------------------------\n");
49. for(temp = head; temp; temp = temp->NextNode)
50. if(!strcmp(temp->name, str)){
51. check = 1;
52. printw("%-20s | %d\n", temp->name, temp->score);
53. }
54. if(!check) printw("search failure: no name in the list\n");
55. }
56. //4-3. 메뉴3: rank번호를 입력받아 리스트에서 삭제
57. else if ( ch == '3') {
58. int num;
59. printw("input the rank: ");
60. echo();
61. scanw("%d", &num);
62. noecho();
63. RankNode \*temp = head;
64. RankNode \*prev = NULL;
65. for(int i = 1; i < num; i++) {
66. prev = temp;
67. temp = temp->NextNode;
68. }
69. if(temp) {
70. if(!prev) {
71. head = temp->NextNode;
72. } else {
73. prev->NextNode = temp->NextNode;
74. }
75. free(temp);
76. printw("result: the rank deleted\n");
77. } else {
78. printw("the rank not in the list\n");
79. }
81. }
82. getch();
83. }

input값으로 1, 2, 3 세가지를 받는다. 1을 입력받는 경우 x에서 y까지 rank를 조회하도록 한다. 이 때 입력으로 아무런 값도 받지 않는 경우 처음부터 끝까지 전부 출력해야 하므로, int x = 1, int y = 10000으로 정의했다. 입력한 값을 화면에 띄워 주기 위해서 아래와 같이 echo() noecho()함수를 호출했다.

echo();

printw("X: ");

scanw("%d", &x);

printw("Y: ");

scanw("%d", &y);

noecho();

받은 x, y 값에 따라서 다음과 같이 전체 LinkedList를 loop로 탐색하면서 해당 범위의 랭크를 띄우도록 한다.

if(x <= y) {

for(i = X; i <= Y; i++) {

if(i >= x && i <= y) {

printw("%-20s | %d\n", temp->name, temp->score);

}

if(i > y) break;

temp = temp->NextNode;

}

} else {

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

}

2를 입력 받는 경우에는 문자열을 입력으로 받아온다. LinkedList를 loop를 돌면서 저장된 이름과 비교하고, 이름에 해당하는 리스트를 모두 출력한다. check 변수로 리스트에 존재여부를 확인하고, 만약 리스트에 이름이 없는 경우, 오류메시지를 출력한다.

3을 입력받는 경우, rank번호를 입력받아서 리스트에서 삭제한다. for문을 돌면서 해당 num까지 LinkedList를 타고 이동한다. 도착한 곳에 노드가 NULL이 아니라면, 해당 노드 전 노드의 nextnode값으로 해당 노드의 nextnode를 연결시켜준다.

3) writeRankfile

void writeRankFile(){

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

int sn, i;

//1. "rank.txt" 연다

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

//2. 랭킹 정보들의 수를 "rank.txt"에 기록

fprintf(fp, "%d\n", rankCount);

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

RankNode \*temp = head;

while(temp->NextNode) {

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

temp = temp->NextNode;

}

fclose(fp);

}

파일을 열어서 업데이트된 rankCount를 적고, temp->NextNode가 Null이 될 때 까지 LinkedList에 있는 노드들의 name, score를 파일에 출력한다. 반복문을 도는 중에 temp = temp->nextnode로 노드사이를 이동하도록 한다.

4) newrank

void newRank(int score){

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

char str[NAMELEN];

int i, j;

clear();

//1. 사용자 이름을 입력받음

printw("input your name\n");

echo();

scanw("%s", str);

noecho();

//2. 새로운 노드를 생성해 이름과 점수를 저장

RankNode \*input\_node = (RankNode\*)malloc(sizeof(RankNode));

strcpy(input\_node->name,str);

input\_node->score = score;

input\_node->NextNode = NULL;

if(head == NULL) {

head = input\_node;

}

else {

RankNode \*curr\_node = head;

RankNode \*prev\_node = NULL;

while(curr\_node != NULL && curr\_node->score >= score) {

prev\_node = curr\_node;

curr\_node = curr\_node->NextNode;

}

//while loop을 돌고난 이후 curr\_node의 위치가 input\_node가 들어갈 위치이다.

if(!prev\_node) {

input\_node->NextNode = head;

head = input\_node;

} else if (!curr\_node) {

prev\_node->NextNode = input\_node;

}

else {

prev\_node->NextNode = input\_node;

input\_node->NextNode = curr\_node;

}

}

rankCount++;

writeRankFile();

}

이 함수는 테트리스가 끝까지 쌓여서 게임이 끝났을 때 사용자의 이름을 입력받아서 rankfile에 이름과 score정보를 저장해주는 기능을 한다.

1. 사용자의 이름을 입력받는다.
2. 새로운 노드를 생성해 이름과 노드를 저장한다.
3. head == null인 경우는 head에 방금 생성한 노드의 주소를 넣어주도록 한다.
4. 아닌 경우 위와 같이 while loop를 돌아 score를 통해 노드가 들어갈 위치를 찾고, 그 자리에 노드를 넣어주도록 한다.
5. rankCount를 1 증가시키고, writeRankFile 함수를 호출해 파일에 업데이트된 내용을 입력한다.

(왜 LinkedList가 여기서 효율적이라고 판단했는지 – 시공간 복잡도 이용 설명)

rank system을 구현하기 위해서 추가해야 하는 기능은 rank.txt 파일의 생성, 조회, 삭제 기능이다. 이 생성, 조회, 삭제 기능을 구현하기 위해서 자료구조를 생각했을 때 고려해야할 부분은 시간 복잡도와 공간 복잡도인데, LinkedList가 차지하는 공간 복잡도는 O(n)이다. 그리고 생성, 조회, 삭제 모두 O(n)의 시간 복잡도를 가진다. array를 사용했을 때도 마찬가지로 O(n)의 시간 복잡도와 공간 복잡도를 가진다. 하지만 array를 사용하는 경우 여분의 공간을 미리 할당해 놔야 하기 때문에 불필요한 공간을 차지하게 된다. 그리고 시간 복잡도의 경우에도 조회시에는 O(1)의 시간 복잡도로 조회할 수 있다는 장점이 있는 반면, 생성 또는 삭제를 하는 경우 원소를 생성 또는 삭제하고 난 이후 배열을 채우거나 밀어주는 작업이 필요하다. 따라서 LinkedList에 비해 추가적인 시간 소모가 이루어진다.

이는 생성과 삭제가 빈번히 이루어지는 rank 시스템에는 어울리지 않는다. 따라서 LinkedList가 rank 시스템을 위한 자료구조로 적합하다는 결론이 나게 된다.

**숙제 구현 결과 첨부**

1. 전체 조회

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

1. KSK의 랭킹정보 불러오기

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

4) 정보가 존재하지 않는 경우

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

5) 3위에 랭크한 ksk정보를 삭제

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

6) 삭제된 것을 확인

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

7)범위를 벗어난 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- 이름을 검색하는 경우 시공간 복잡도. : 이름을 검색하는 경우 모든 노드를 조회하는 것이 필수이므로 O(n)만큼의 시간복잡도를 가진다. 공간복잡도는 O(1)이다.

- 삭제 알고리즘 :

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**2. 습득한 내용**

테트리스의 랭킹 시스템을 구현해보는 활동을 통해 프로그램의 의도와 작동원리에 따라 시공간 복잡도를 최대한 낮출 수 있는 알고리즘을 생각해보고, 적용하는 일련의 과정을 거치면서 전체적으로 프로그램의 알고리즘을 짜는 계획 및 설계 활동을 경험해볼 수 있었고, 그 과정에서 자료구조 선택의 중요성을 제고하는 기회가 됐다.

이 과정에서 LinkedList가 어떤 상황에서 유용하게 쓰일 수 있는 지 array는 어떤 상황에서 유리한 지를 생각해보는 시간을 가졌다. LinkedList가 유리한 상황은 이번 rank 구현 프로젝트와 같이 시간보다는 공간을 효율적으로 활용하는 것이 중요할 때임을 알게 되었다.

LinkedList를 자료구조로 사용하는 경우에 array보다 조회하는 속도는 느리지만, 생성, 삭제 등의 작업을 하는 데는 유리한 것을 알 수 있었다. array의 조회 시간 복잡도는 O(1)이고, LinkedList의 조회 시간 복잡도는 O(n)이다. array는 직접 주소로 접근하지만, LinkedList는 노드를 거쳐 해당하는 위치까지 도달해야 하기 때문이다. 하지만 생성, 삭제 등의 작업을 하는 경우, LinkedList는 생성 또는 삭제 이후 전의 노드와 다음 노드를 포인터로 연결해주기만 하면 작업이 완료되는 반면, array의 경우 뒤에 위치하는 모든 원소들의 위치를 다시 조정해주는 작업이 더 필요하기 때문에 시간 복잡도 면에서 불리함을 알 수 있었다. 생성 또는 삭제를 할 때 둘 다 시간 복잡도는 O(n)으로 표기되지만, 디테일한 면에서 이러한 차이를 확인할 수 있었다.