**CSE3013 컴퓨터공학 실험: 테트리스 프로젝트**

분반: 1반 학번: 20171697 이름: 최민영

1. **설계 문제 및 목표**

ncurses 라이브러리가 제공하는 리눅스 터미널 상에서의 GUI를 이용하여 테트리스 게임 프로그램을 제작하고 여기에 랭킹 시스템과 블록 배치 추천 기능을 추가 구현한다.

테트리스 게임은, 블록을 90도씩 반시계 방향으로 회전하거나 세 방향(좌, 우, 하)으로 움직여 필드에 쌓으면, 빈틈없이 채워진 줄은 지워지고 그에 따른 스코어를 얻어 결과적으로 가장 높은 최종 스코어를 얻는 것이 목적이다.

프로그램을 실행하면 사용자는 메뉴를 선택할 수 있다. 1번을 선택하면 테트리스 게임을 플레이할 수 있고, 2번을 선택하면 랭킹 정보를 확인할 수 있고, 3번을 선택하면 블록 배치 추천 기능을 따라 플레이되는 모드로 테트리스 게임이 실행되고, 마지막으로 4번을 선택하면 프로그램이 종료된다.

1주차에는 테트리스 게임을 돌아가게 하는 play함수와 blockdown함수를 구현하여야 했다. 이 때 구현해야할 상황은 아래와 같았다.

1. 1초마다 블록이 떨어지고 입력받은 키에 맞게 블록이 움직이어야 한다.
2. 블록이 떨어질 위치가 그림자로 표현되어야한다.
3. 블록이 field에 쌓이면 field와 지금 쌓아진 블록의 아래면이 만나는 면적을 계산해 score을 증가시켜야 한다.
4. 블록이 쌓인 후 field에서 한 줄이 모두 차있으면 그 줄을 삭제하고 이렇게 삭제되는 줄의 개수에 따라 score을 증가시켜야 한다.

2주차에는 테트리스 게임의 랭킹시스템을 구현해야 했다. 해야할 일은 다음과 같았다.

1. 랭킹이 저장되어있는 rank.txt 파일을 읽어서 큰 점수 순서대로 랭킹시스템 자료구조를 만든다.
2. 테트리스 게임이 game over로 종료되면 사용자의 이름을 입력받아 기존의 랭킹에 추가해준다.
3. 알고싶은 랭킹 범위를 입력받아서 입력받은 범위의 랭킹을 출력한다.
4. 랭킹을 입력받고 해당 랭킹을 삭제한다.
5. 이름을 입력하면 해당 이름으로 저장된 점수를 큰 순서대로 출력한다.
6. 프로그램을 끝낼 때 현재 랭킹시스템 자료구조를 txt파일로 저장하도록 한다.

3 주차에는 블록을 놓을 위치를 추천해주는 시스템과 추천 위치로 블록을 쌓아가는

자동플레이 모드를 만든다. 대략적인 방법은 아래설명과 같다.

1. 현재 상황을 root로 다음으로 떨어지는 블록이 쌓일 수 있는 경우의 수만큼 child node가 만들어지는 트리를 만든다. 이 때 트리의 높이는 고려하는 블록의 개수이다. 만들어진 트리의 leaf node 중 가장 점수가 큰 노드를 파생시키는 다음 블록의 위치로 추천위치를 정한다.
2. 위 시행을 블록을 떨어뜨릴 때마다 행하고 추천 블록은 R로 표시를 해준다.
3. 자동플레이 모드의 경우, 사용자로부터 q 또는 Q로 게임을 종료시키는 키만 받고 위의 경우보다 더 효율적인 prunning 방법으로 트리를 구성해서 추천위치를 구해낸다.
4. **요구사항**
   1. 설계 목표 설정

**Play**

|  |  |
| --- | --- |
| play() | InitTetris로 게임시작 환경을 만들고 ProcessCommand로 이용자의 키 입력에 맞는 명령을 수행하는 과정을 q,Q가 입력되거나 gameover flag가 켜질 때까지 한다. 게임이 끝나는 환경에 맞게 DrawBox를 이용해 네모난 칸에 알맞은 메시지를 출력한다. |
| InitTetris() | 다음에 올 블록, 떨어질 블록의 상태(회전수, 위치), 점수를 초기화하고 gameover flag를 끈다. field를 비우고 DrawOutline, DrawField, DrawBlock, DrawNextBlock, PrintScore으로 화면을 초기화시킨다. |
| DrawOutline()  DrawField  DrawBlock  DrawNextBlock  PrintScore  DrawBox | field 테두리와 nextblock이 보여지는 테두리, score 보여지는 테두리를 그린다. / field에 저장된 대로 출력한다/ 인자로 받은 위치에 입력받은 블록의 id와 회전수에 따라 블록을 인자로 받은 character로 출력한다/ 다음 블록이 그려지는 곳에 다음으로 떨어질 블록을 출력한다 / 점수칸으로 주어진 곳에 점수를 출력한다 / 인자로 받은 위치에 인자로 받은 크기에 맞게 네모난 칸을 그린다. |
| Getcommand() | 입력받은 키에 알맞은 명령을 반환한다. |
| ProcessCommand() | 블록이 명령에 맞게 움직일 수 있는 지 CheckToMove로 확인한 후 움직일 수 있으면 바뀐 블록이 화면에 알맞게 나타나도록 DrawChange를 호출한다. |
| CheckToMove() | 인자로 받은 위치에 인자로 받은 원하는 블록을 그릴 수 있는 지 알아보는데, 블록을 그릴려는 곳이 이미 field에 차있거나 field를 벗어나면 0을 반환한다. 그렇지 않으면 1을 반환한다. |
| DrawChange() | 명령을 받고 바뀌기 전 블록의 위치를 파악하여 지우고 새로운 블록을 그린다. |
| DrawBlockWithFeatures() | 인자로 받은 블록의 위치에 블록을 그리고 DrawShadow함수로 그에 맞는 그림자를 동시에 그린다 |
| DrawShadow() | CheckToMove로 더이상 내려가지 않는 위치를 찾아서 그 위치에 블록을 ‘/’으로 그린다. |
| BlockDown() | 1초마다 호출되는 함수이고 아래로 더 움직일 수 있으면(CheckToMove로 확인) 블록의 위치를 아래로 한 칸 움직이게 하고, 더 이상 아래로 움직일 수 없을 때 블록이 초기 위치와 같으면 gameover임을 판단하고 그렇지 않다면 블록을 쌓고 지울 수 있는 줄을 지워서 점수 증가시킨다. 다음 블록을 떨어뜨리기 위한 블록에 대한 정보를 초기화하고 nextBlock을 새로이 채워준다. |
| AddBlockToField() | 인자로 받은 블록의 위치와 회전 수, id에 맞게 field에 블록을 저장한다. 이 때 field와 인자로 받은 블록의 바닥면과 만나는 면적을 계산해서 추가되는 점수를 반환한다. |
| DeleteLine() | field를 살피면서 한 줄 모두 차있는 줄이 있으면 지우고 그 위로 쌓여있는 블록을 모두 한 칸씩 내린다. 지워지는 줄의 수를 세서 그에 따른 추가되는 점수를 계산해서 반환한다. |

**rank**

|  |  |
| --- | --- |
| createRankList() | rank.txt에 쓰여있는 정보를 생각해낸 자료구조에 맞게 저장해둔다. |
| rank() | 사용자의 입력에 맞게 함수를 호출한다 |
| findbyname() | 이름을 입력받아서 저장되어있는 랭킹 자료구조에서 가장 큰 점수부터 살펴보면서 해당 이름과 같은 정보가 있으면 점수를 출력한다. |
| displayrank() | 입력받은 랭킹의 범위에 맞게 그 범위에 있는 사용자 이름과 점수를 출력한다. |
| deleterank() | 랭킹을 입력받아서 랭킹에 해당하는 정보를 랭킹시스템에서 삭제한다. |
| newRank() | 새로운 정보를 입력받아서 랭킹시스템에 추가한다. |
| writeRankFile() | 랭킹 시스템 자료구조에 저장된 내용을 rank.txt 파일에 써서 저장하고 랭킹시스템 구현을 위해 동적할당된 메모리를 free시켜준다. |

**recommended play**

|  |  |
| --- | --- |
| recommend() | 인자로 들어온 블록이 쌓여있는 상태, 점수, 깊이를 통해서 다음 블록을 쌓을 수 있는 모든 경우를 인자로 들어온 블록의 child 노드로 만든다. 이렇게 만들어진 모든 경우에 대해서 또 다음 블록이 쌓일 수 있는 경우를 따져보기 위해, 모든 경우에 대해 재귀적인 함수 호출을 한다. 예상해볼 블록 개수만큼 깊이가 만들어지면 점수를 반환한다. 반환된 점수가 가장 큰 지금 떨어질 블록의 위치를 전역변수에 저장한다. |
| del\_recnode() | 동적할당으로 만들어놓은 추천블록 찾기 위한 트리를 free시켜준다. |

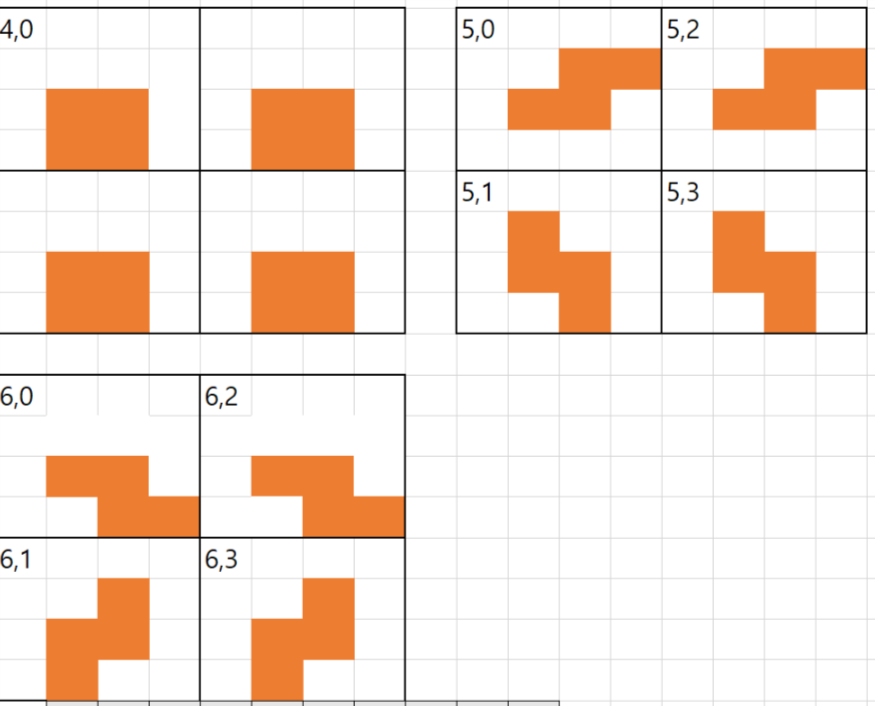
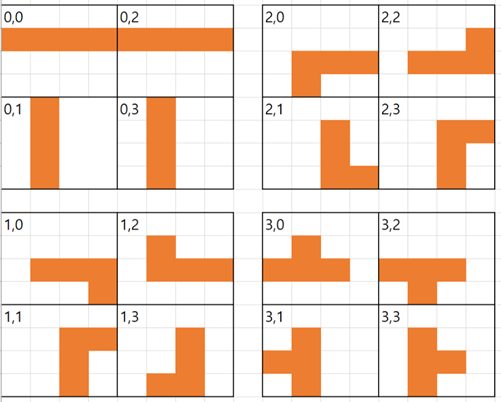
|  |  |
| --- | --- |
| recommendedPlay() | 사용자로부터 q 또는 Q 키를 받아서 프로그램을 종료시키고 modified\_recommend로 추천받은 블록 위치로 블록을 자동으로 쌓는다. |
| Init\_recommend() | 새로운 블록을 떨어뜨리기 전에 항상 호출되어 떨어질 블록의 쌓일 위치를 recommendR, recommendX, recommendY에 저장하는 역할을 한다. 추천위치를 찾는 것은 실제로 modified\_recommend로 이 함수를 부르기 전에 현재 field정보를 구조체에 맞게 만드는 작업을 수행하고, 추천시스템 트리를 만드는 데 걸리는 시간을 측정한다. |
| modified\_recommend() | recommend()함수처럼 인자로 들어온 블록이 쌓여있는 상황에서 다음 블록을 쌓을 수 있는 경우의 수를 고려해서 고려할만 하면 이를 인자로 재귀적으로 함수를 호출해서 이 경우에 대한 다음 블록을 쌓는 경우를 고려하도록 한다. 다음으로 살펴볼 블록 개수만큼 깊이가 만들어지면 재귀적 함수 호출을 멈추고 최대 점수가 반환되는 지금 떨어지고 있는 블록의 쌓이는 위치를 전역변수에 저장한다. |

* 1. 합성
* **자료구조**

1. *Play를 위한 자료구조*

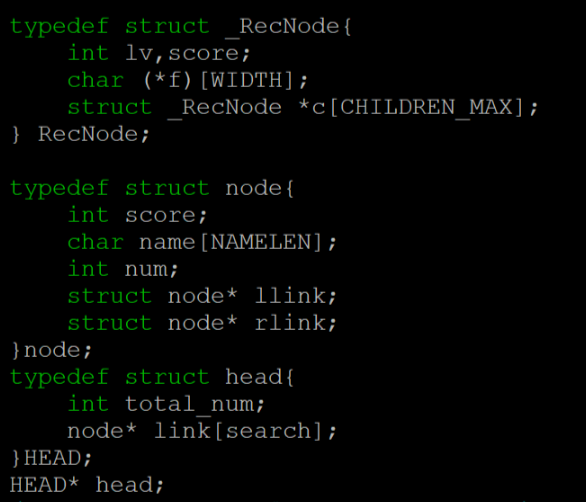
field 정보가 22개의 행과 10개의 열로 구성된 2차원 배열에 저장되어 있다. 블록으로 차있으면 값이 1로, 채워져있지 않으면 0으로 저장되어있다.

블록의 정보는 전역변수로 (블록 id, 블록 회전 수)에 따라 다음과 같이 4\*4 2차원배열로 저장되어 있다. (블록이 있는 곳은 1, 블록이 없는 곳은 0으로 저장되어 있다)

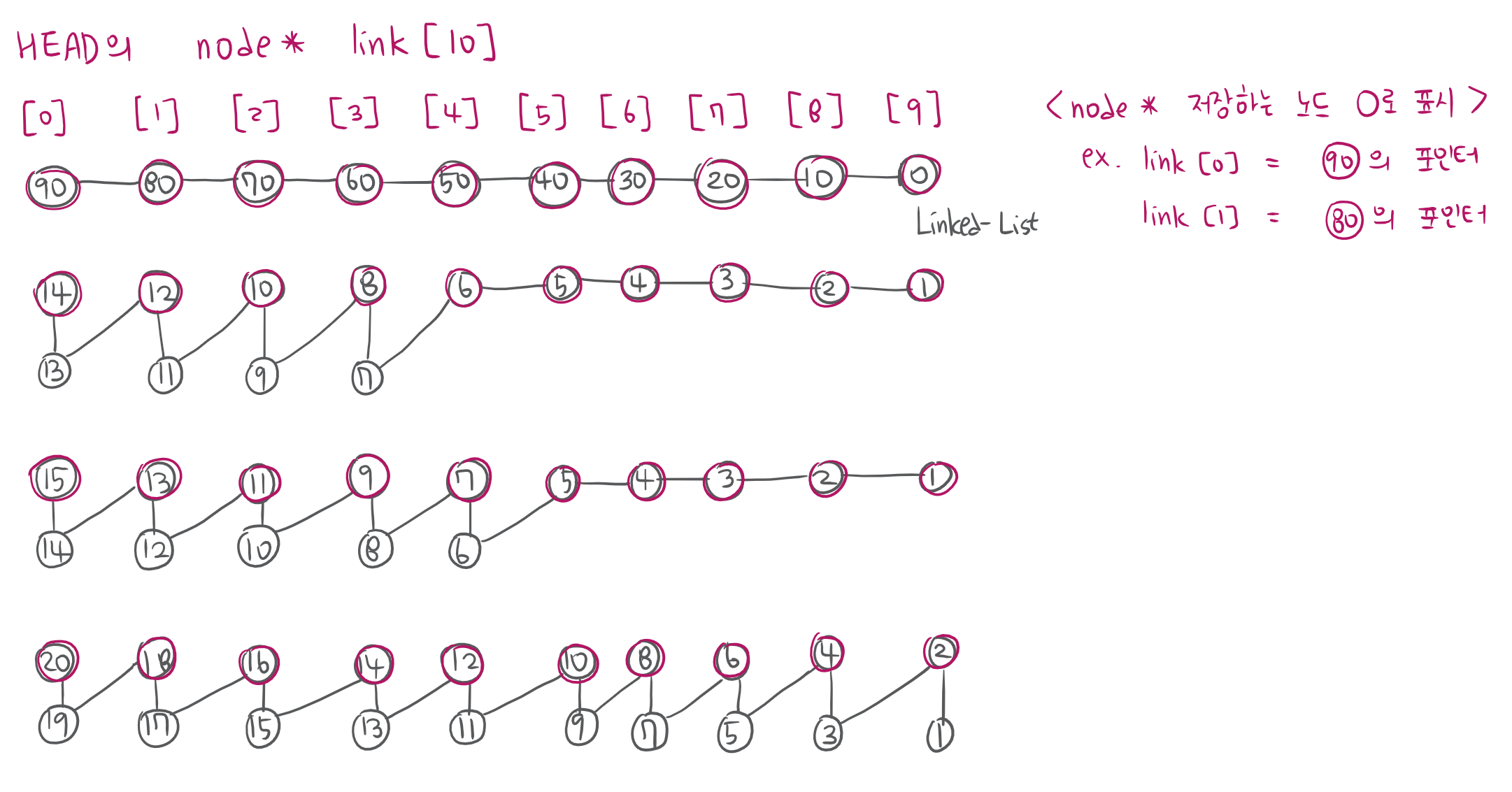


1. *rank를 위한 자료구조*

점수가 큰 순서대로 연결되어 있는 Linked-List를 구현했으며 좀 더 빠른 탐색을 위해 head라는 구조체를 만들고 그 안에 Linked-List를 구성하고 있는 노드의 pointer 배열을 만들었다. 이 배열의 용도는 배열의 크기 만큼 Linked-List의 중간 노드들을 접근할 수 있게 하기 위한 것이다. 즉, 배열과 달리 linked-list의 경우 중간을 참조할 수 없다는 단점을 해소하고자 하였다. 예를 들어 아래 그림과 같이 구현되도록 했다.



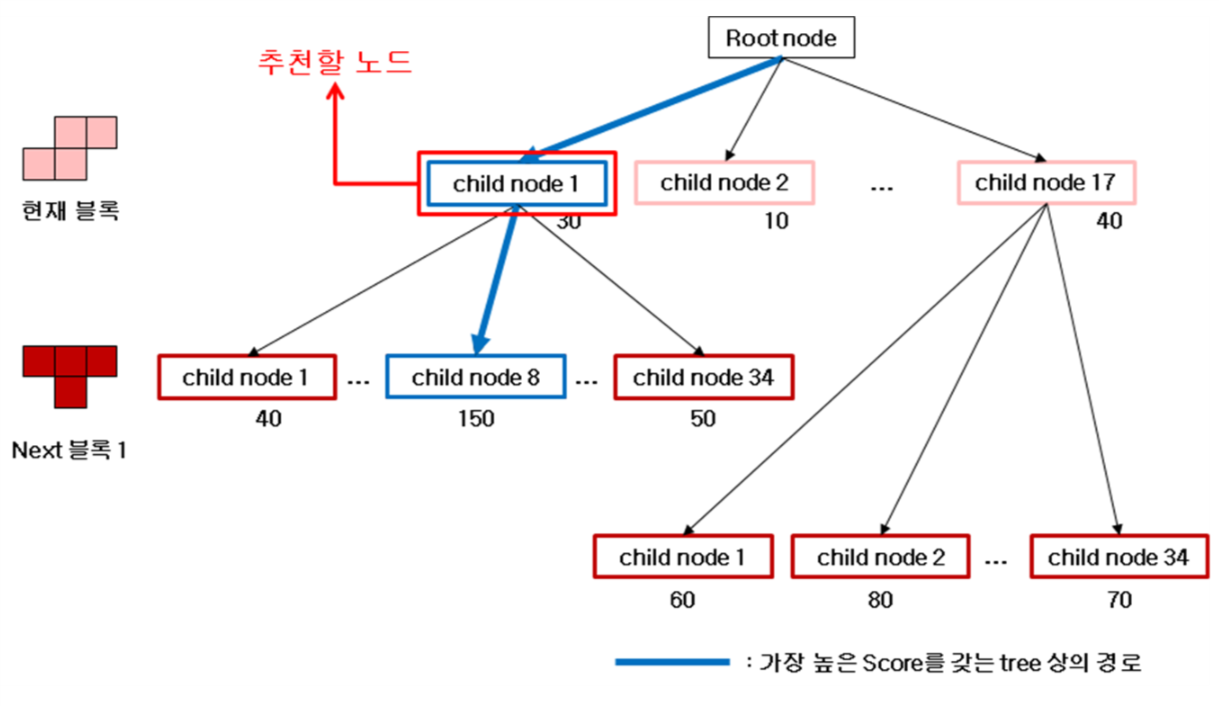
<search는 사용자 지정 상수이다(#define search 10) >



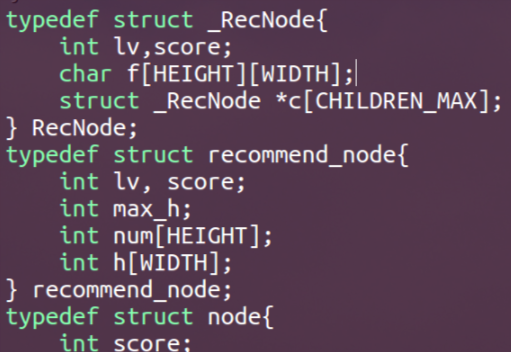
이렇게 자료구조를 생성하면, 탐색과 삽입 삭제 시에 head->link로 연결된 노드들부터 살펴보면 더 시간적으로 효율적이라는 생각을 하였다.

1. *recommend를 위한 자료구조*

공통적으로는 트리를 형성한다.



1. recommend 위치를 play에서 구현하는 경우



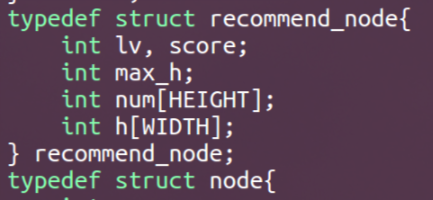
lv는 nextBlock의 몇 번째 블록을 수행하고 난 결과를 저장하고 있는 지 알려주는 변수

score은 지금까지의 누적 점수

f[HEIGHT][WIDTH] lv번째 블록까지 사용한 테트리스 field를 저장하고 있는 2차원 배열

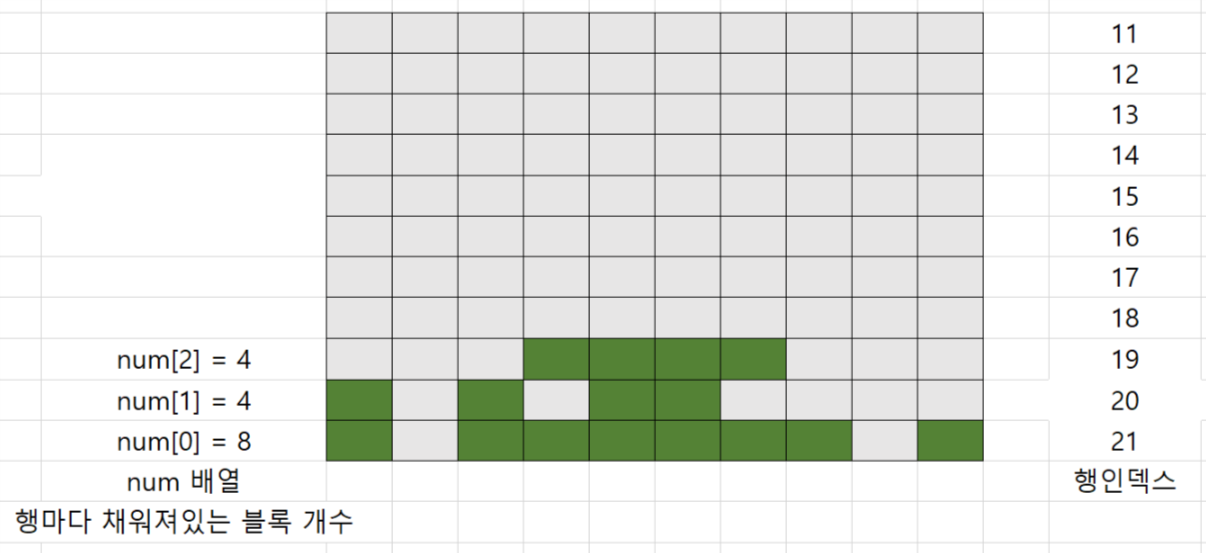
c배열은 이 구조체의 정보를 바탕으로 lv+1번째 블록을 쌓을 수 있는 결과 노드들의 주소들을 담고 있다.

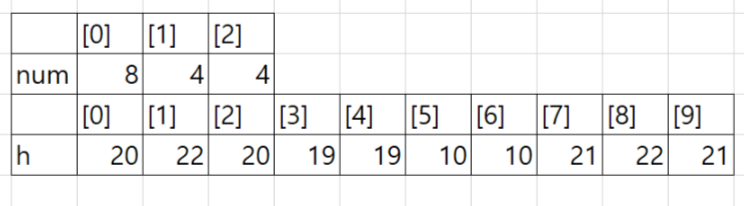
1. recomend 위치대로 자동 플레이하는 경우



이 구조체는 play 중인 field에서 nextBlock의 lv번째 블록까지 쌓아본 결과를 담고 있다. 즉 lv는 지금까지 쌓아본 블록이 nextBlock의 몇 번째까지인 지를 담고 있다. 그리고 그 쌓아본 결과 점수를 score변수가 담고 있고 field정보를 max\_h, num, h가 담고 있다. 지금까지의 쌓여진 최대높이를 max\_h에 저장하고, num배열은 높이마다 블록이 차 있는 개수를 저장하고 있고 h는 열마다의 블록이 처음으로 존재하는 행 인덱스를 저장하고 있다.

예를 들면,





이렇게 저장해두었다.

그리고 이러한 구조체를 생각한 이유는, h배열로 블록이 쌓일 수 있는 곳을 찾을 수 있고 num배열로 블록을 쌓으면 한 행이 지워지는 지 아닌 지를 판단할 수 있기 때문이다. 즉, 블록을 쌓고 나서 어떤 행을 채우고 있는 칸의 개수가 WIDTH와 같아지면 한 줄이 모두 채워진 것으로 지울 수 있음을 판단할 수 있따. 그래서 블록이 어디에 쌓이고 이 때 점수는 어떻게 될 지를 판단할 수 있다. 그렇지만 이 경우 예상의 오차가 없으려면 칸으로 둘러쌓인 공백이 없어야한다. 위 예시를 보면 밑에서 두 번째 행에 4번 째 칸에 공백이 있는 데 이 경우 밑에서 세번째 행이 지워진다면 위 구조체 정보로는 h[3]가 1 늘어나는 것으로 생각해 이 공백이 차있는 것으로 생각하고 블록을 쌓게 된다. 그렇게 되면 블록이 쌓이는 위치에서 오차가 발생할 수 있다. 이러한 오차때문에 현재 field상태를 root로 nextBlock의 BLOCK\_NUM번째까지 예상한 BLOCK\_NUM만큼의 깊이 트리를 만들고 최대 점수를 얻는 경로를 찾아 그 경로대로 진행할 수 없다. 따라서 최대 점수를 얻을 수 있게 해주는 다음 블록의 위치만을 알아내고 다시 그 블록을 쌓은 후의 field정보를 root로 한 트리를 다시 만들어야한다. 따라서 나는 트리구조를 만들지 않고 재귀함수를 통해서 최대점수를 얻게 해주는 다음 블록이 쌓일 위치를 전역변수에 저장하는 방식을 취했다.

* **알고리즘**

1. *play를 위한 알고리즘*

**CheckToMove()**

2중 루프로 모든 행의 모든 칸을 살펴본다.

block[currentBlock][blockRotate][i][j] == 1일 때 i+blockY와 j+blockX가 field 범위를 넘어가는 지 판단하고 또는 field[i+blockY][j+blocX] =1인 지 판단해서 그렇다면 0을 그렇지 않으면 1을 반환한다.:

**DrawChange()**

현재 블록 정보와 블록이 바뀐 command를 인자로 받아 블록이 바뀌기 전 위치나 모양을 알아낸다. switch를 이용하여 command에 따라 다르게 찾아낸다. 이전 블록을 DrawBlock에 인자 ‘ . ‘을 보내는 방법으로 지우고, 현재 블록을 DrawBlock에 인자 ‘ ‘을 보내서 그린다.

**AddBlockToField()**

2중 루프로 block의 모든 행의 모든 칸을 살펴보면서 Block[currentBlock][blockRotate][i][j]가 1이면 field[blockY+i][blockX+j]를 1로 바꾸어주었다.

**DeleteLine()**

가장 위의 행(i=0)부터 모든 칸을 살펴보면서 field[i][j]가 처음으로 1이면 이 때의 행을 Max\_Height 변수의 값으로 저장한다. 그리고나서 이 바닥 행부터 Max\_Height 변수 값까지모든 칸을 살펴보면서 어떤 행의 모든 칸이 채워져있으면 해당 행의 위부터 Max\_Height까지 모두 아래로 옮기고 Max\_Height를 감소시킨다.

**BlockDown()**

CheckToMove함수에 y+1을 인자로 보내 한 칸 내려갈 수 있는 지 확인하고 된다면 y좌표를 증가시켜준다. 내려갈 수 없을 때 y좌표가 -1로 초기상태에서 내려가지 못한 경우라면 gameOver 변ㄴ수를 1로 바꾼다. 그렇지 않았다면 AddBlockToField와 DeleteLine 함수를 호출해 블록을 필드에 쌓은 결과를 저장하고 점수를 알아낸다. nextBlock 재지정해주고 blockRotate, blockY, blockX를 초기화해준다. DrawNextBlock, PrintScore, DrawField, DrawBlock을 호출한다.

**DrawBlock()**

2중 루프로 블록의 한 칸씩 살피면서 block[blockID][blockRotate][i][j] =1이면 move blockY+i, blockX+j로 커서를 옮긴 후에 인자 tile을 출력한다.

**DrawBlockWithFeatures()**

인자로 들어온 블록을 출력하기 위애 DrawBlock을 호출하고 그에 따른 그림자를 그리기 위해 DrawShadow를 호출한다.

**DrawShadow()**

CheckToMove를 이용해 현재 블록 위치에서 더 이상 내려갈 수 없는 blockY좌표를 구한다. 구하면 그 위치에 DrawBlock에 ‘/’을 인자로 보내 그리게 한다.

1. *rank를 위한 알고리즘*

**createRankList()**: txt파일로 저장되어있는 랭킹을 Linked-List로 구현하는 함수

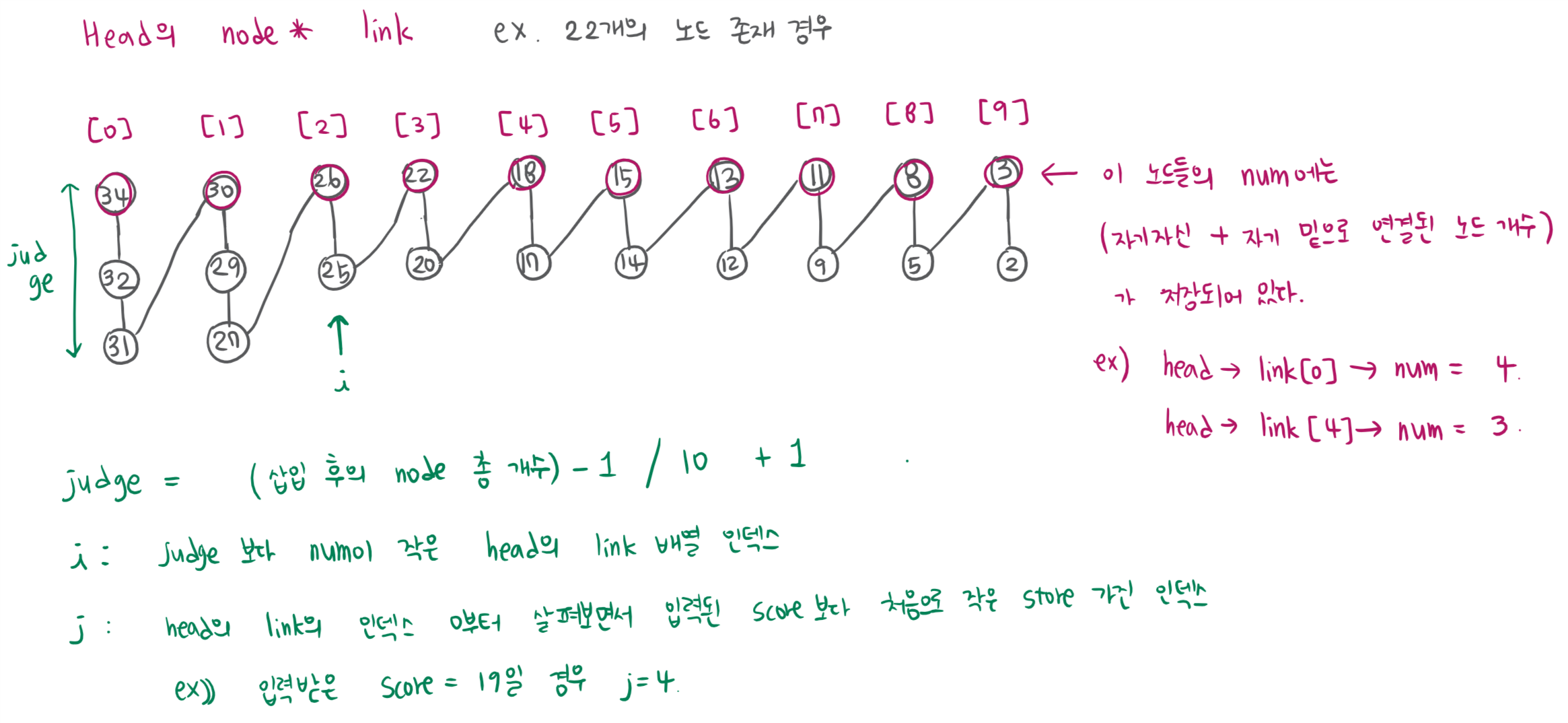


위에서 말한 자료구조를 만들기 위해서는 위의 그림을 참고로 보면, b인덱스까지는 per번째 노드마다 head->link배열에 넣어주고 b인덱스부터는 (per-1)번째 노드마다 head->link배열에 넣어주면 된다. ( 34개의 노드가 있을 경우로 예를 들면, 인덱스 4까지는 4번째 노드마다 head->link배열에 넣어주고 인덱스 4부터는 3번째 노드마다 head->link배열에 넣어준다)

노드의 총 개수가 10보다 같거나 작을 경우에는, 노드 개수만큼 파일에 쓰여있는 정보를 저장하고 있는 노드를 만들고, 랭킹 순대로 연결하고, 모든 포인터를 head->link 배열에 넣는다.

노드의 총 개수가 10보다 큰 경우에는, 파일 정보를 저장한 노드를 모두 만들고 랭킹순대로 연결한 뒤에, 위에서 설명한 대로 b인덱스 전까지는 per번째마다 head->link에 넣어두고 b인덱스 후에는 (per-1)번째마다 head->link에 넣어주었다.

**newRank(int score)**: score을 가지는 새로운 노드를 Linked-List에 추가하는 함수



삽입하기의 과정)

j가 0인 경우에는 새로운 노드를 linked\_list 맨 앞에 삽입하는 방법을 따른다.

j가 0이 아닌 경우에는 새로운 노드를 삽입할 앞의 노드가 존재하는 경우로, head->link[j-1]이 가리키는 노드부터 살펴보면서 삽입할 위치를 찾고 삽입한다.

head->link 값 변경하기)

삽입 후의 node 총 개수가 10보다 작은 경우에는 k=j로 시작하고, tmp는 새로운 노드 포인터 값으로 시작해서 head->link[k]에 tmp를 저장하고 tmp를 다음 노드(tmp->link)로 바꿔가면서 실행하다가 tmp가 NULL이 되면 중단한다.

삽입 후의 node 총 개수가 10보다 크거나 같은 경우에는, 여러 경우를 살펴본 결과 2가지로 일반화 되었다.

(살펴본 경우:

j-1은 삽입할 노드가 그 밑으로 연결될 head->link의 인덱스

i는 head->link의 인덱스값으로, 삽입할 경우 이 인덱스 밑으로 연결된 노드가 하나 더 생겨야하는 인덱스를 의미한다

이럴 때, (j-1)이 0인 경우에 i가 0일 경우 또는 그렇지 않을 경우 /

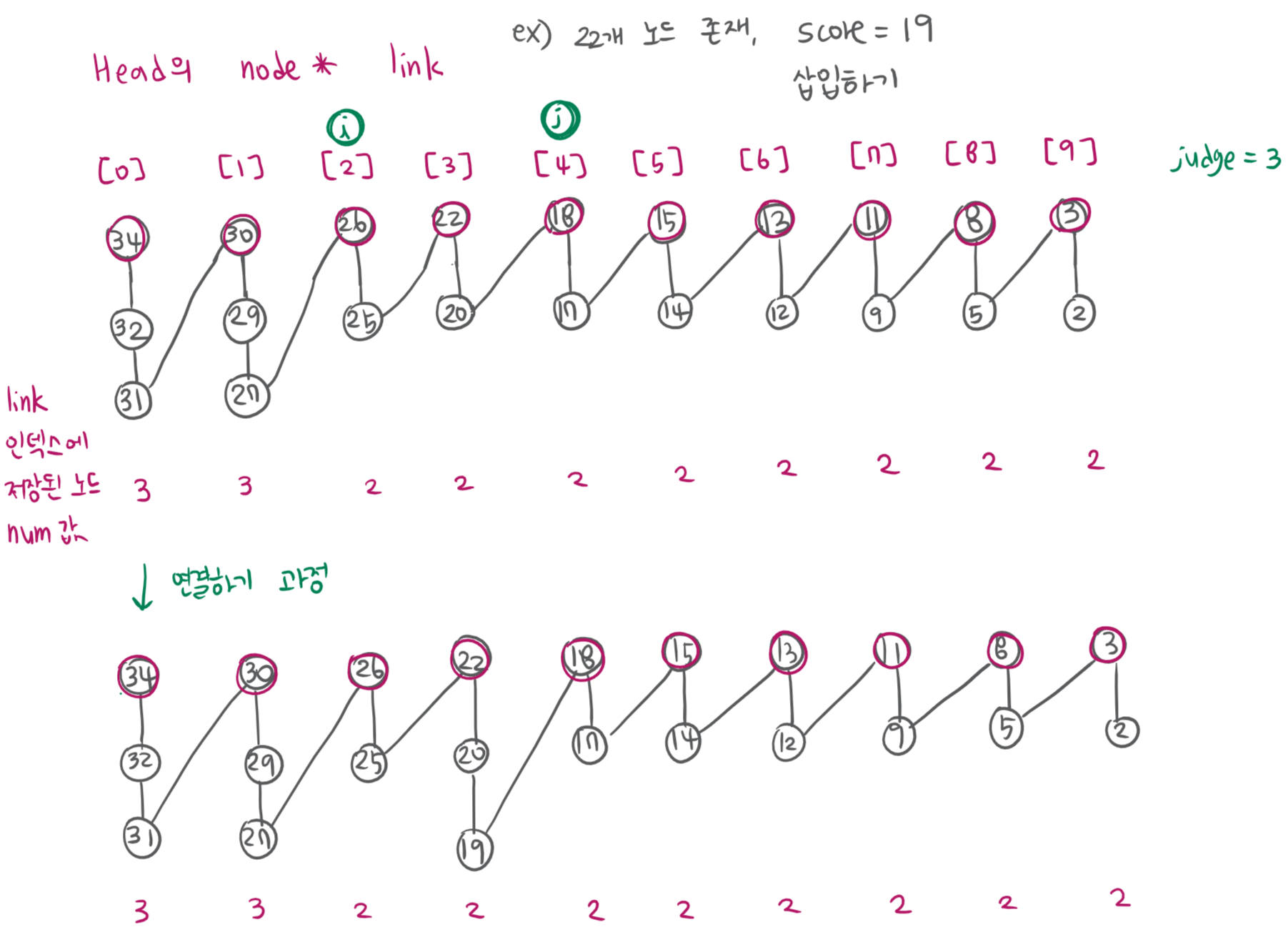
(j-1)이 9일 경우에 i가 9일 경우와 그렇지 않을 경우 /

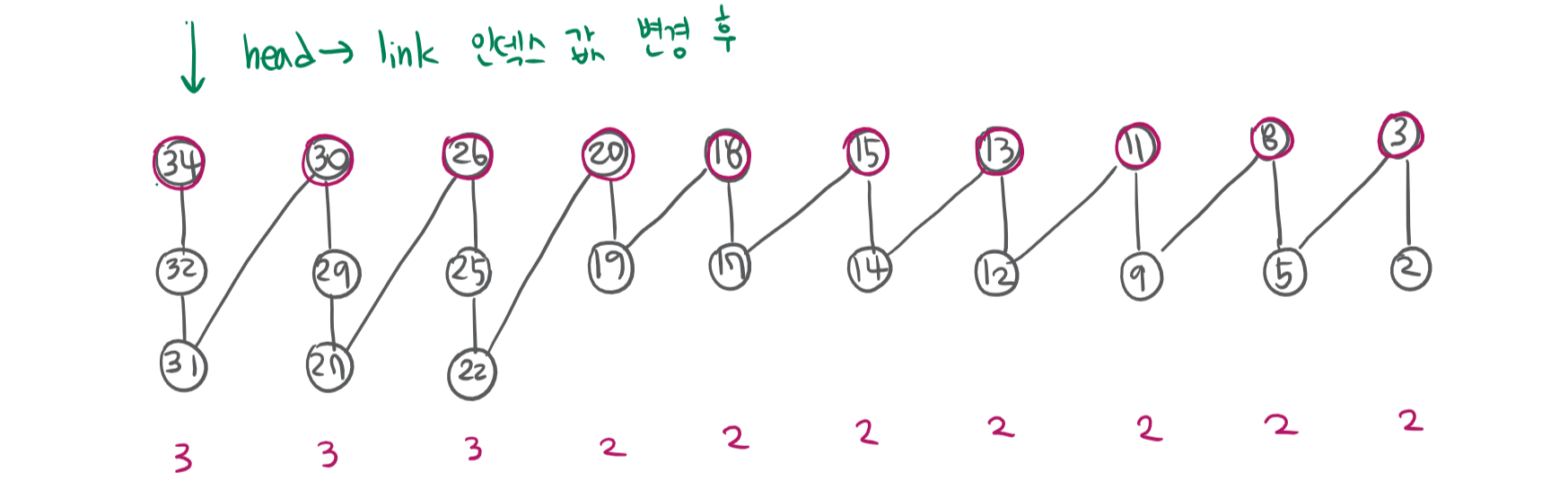
(j-1) < i 경우 / (j-1) = i 경우 / (j-1) > i 경우 )

1. (j-1) < i : head->link 인덱스 i+1부터 j-1까지 인덱스에 저장된 값을 기존 head->link가 가리키던 노드의 오른쪽 노드 포인터로 변경한다. 이 때 노드의 num값도 바꿔준다.
2. (j-1) > i : head->link 인덱스 j부터 i까지 인덱스에 저장된 값을 기존 head->link로 연결되었던 노드의 왼쪽 노드 포인터로 변경한다. 이 때 노드의 num값도 바꿔준다

두 가지를 수행한 후에 head->link[i]->num 값을 judge로 바꾼다.

삽입과정의 예)

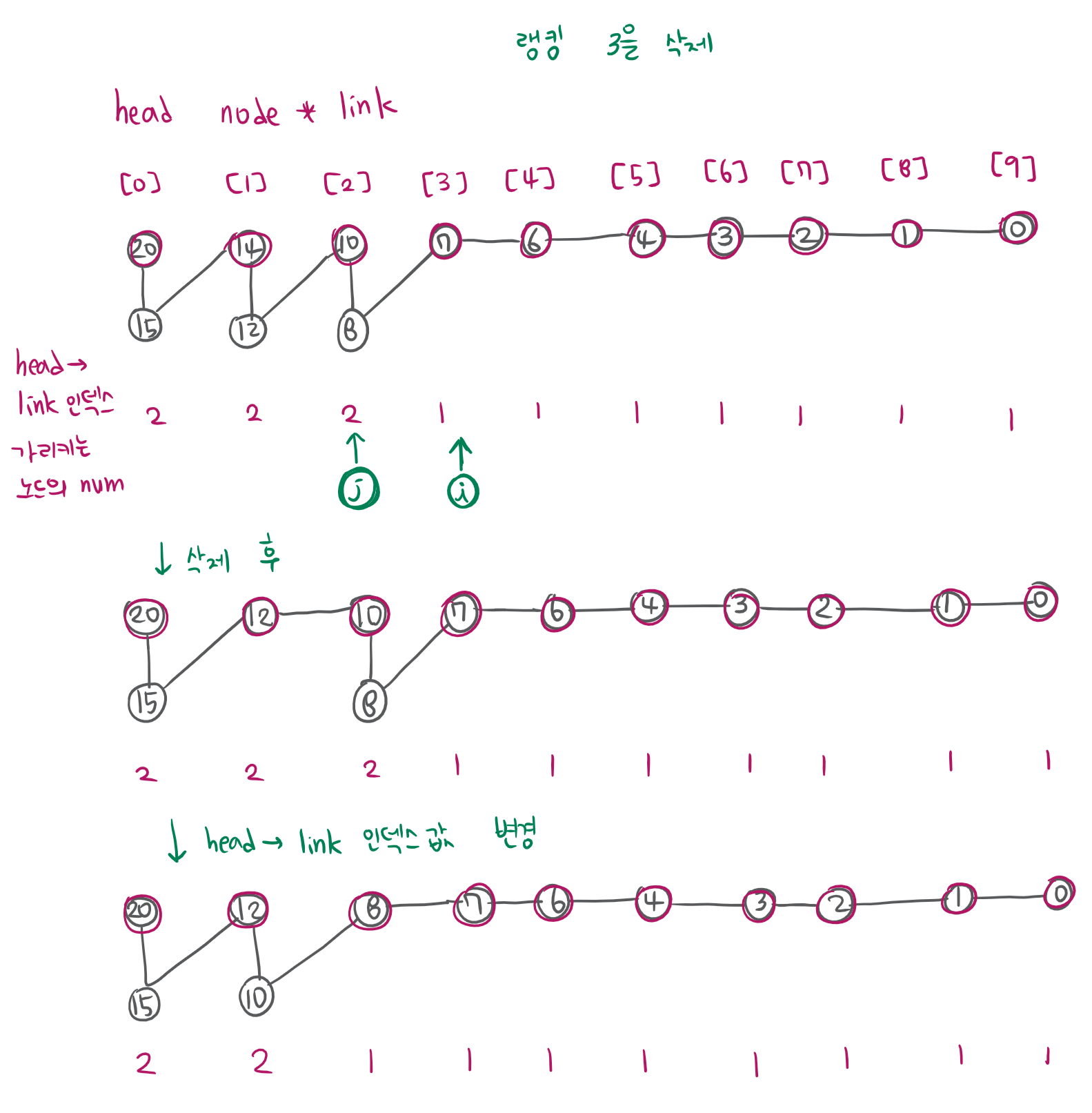


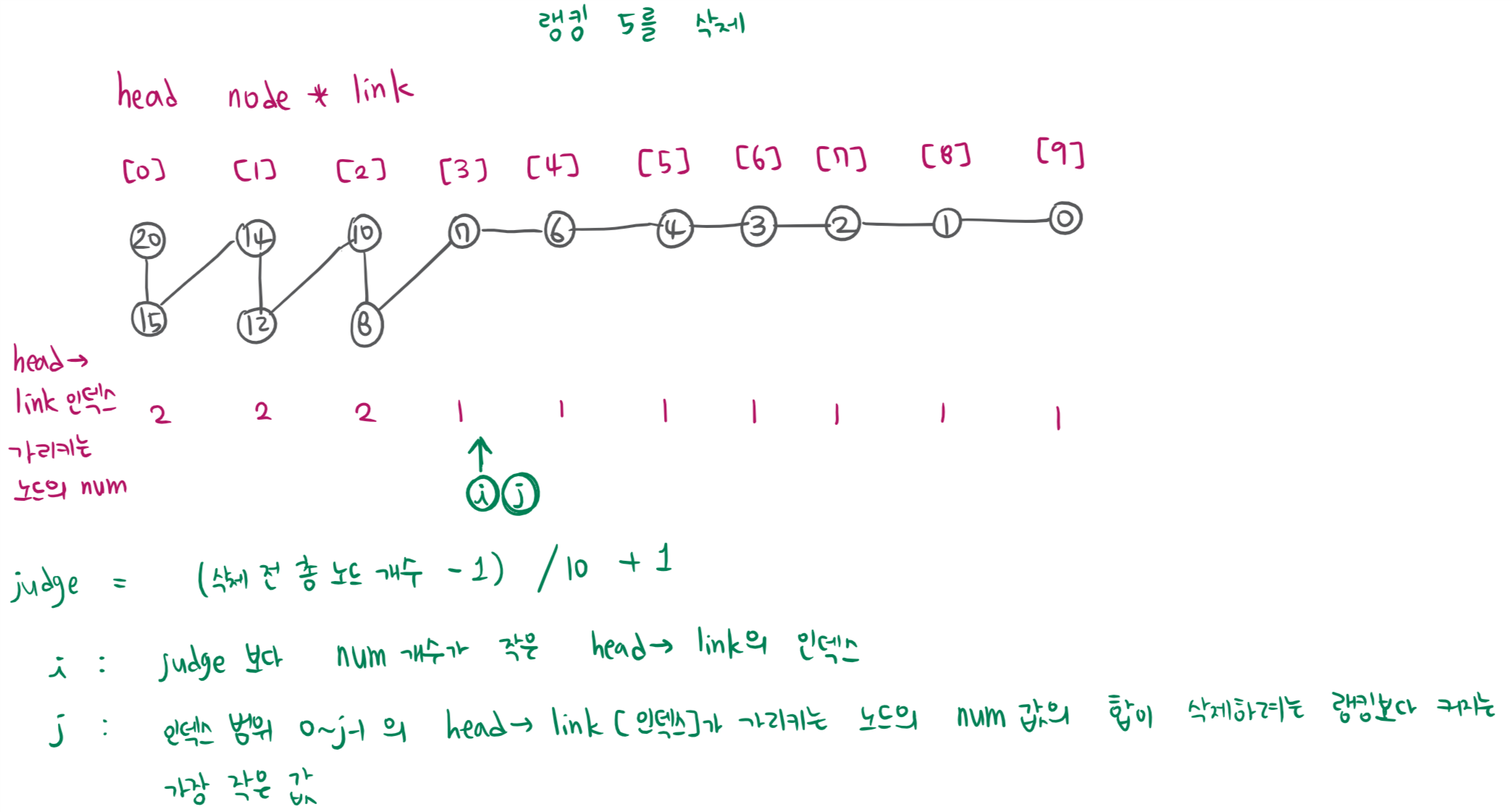


**deleterank()**: 입력받은 랭킹의 노드를 Linked-List로부터 삭제하는 함수

현재 노드 총 개수가 10보다 작은 경우) 삭제할 랭킹을 del이라고 하면 head->link[del-1]의 전 노드와 다음 노드를 연결한 후 head->link[del-1]을 삭제한다. 그리고 나서 cur = head->link[del], 인덱스 값을 del-1부터 시작해서 head->link[해당 인덱스]를 cur값으로 저장하고 cur을 다음 노드로 바꾸는 수행을 cur이 NULL이 될 때까지 수행한다.

현재 Linked-List의 총 노드 개수가 10보다 클 경우) 다음 그림과 같이 j와 i를 설정한다.





j와 I 구하는 과정)

모든 head->link를 살피면서 i와 j에 해당하는 조건에 걸리면 해당 인덱스로 i와 j를 설정해준다.

노드 삭제 과정)

삭제하려는 노드가 head->link에 저장되어 있는 경우에는 head->link의 해당 인덱스 값에 삭제할 노드 대신에 삭제할 노드 다음 노드 포인터를 저장한다. 삭제하려는 노드가 head-> link에 저장되어 있지 않은 경우라면, j-1은 삭제하려는 노드가 밑으로 연결되어 있는 head->link의 인덱스이고 cur=head->link[j-1]인 cur은 num+1번째 랭킹이다. cur이 del번 째 랭킹을 가리키도록 한 뒤에 cur의 전 노드와 다음 노드를 연결한 후 cur을 free시킨다.

head->link의 인덱스 값 변경하기)

여러가지 경우를 따져본 결과 방법은 2가지 경우로 일반화할 수 있었다.

(살펴본 경우:

(j-1)은 삭제한 노드가 head->link[j-1]이 가리키는 노드였거나 그 밑으로 연결되었던 head->link의 인덱스 값이다. (i-1)은 head->link[i] 밑으로 연결되는 노드 개수가 줄어야할 인덱스값이다.

(j-1)이 (i-1)보다 작은 경우/ (j-1)이 (i-1)과 같은 경우 / (j-1)이 (i-1)보다 큰 경우

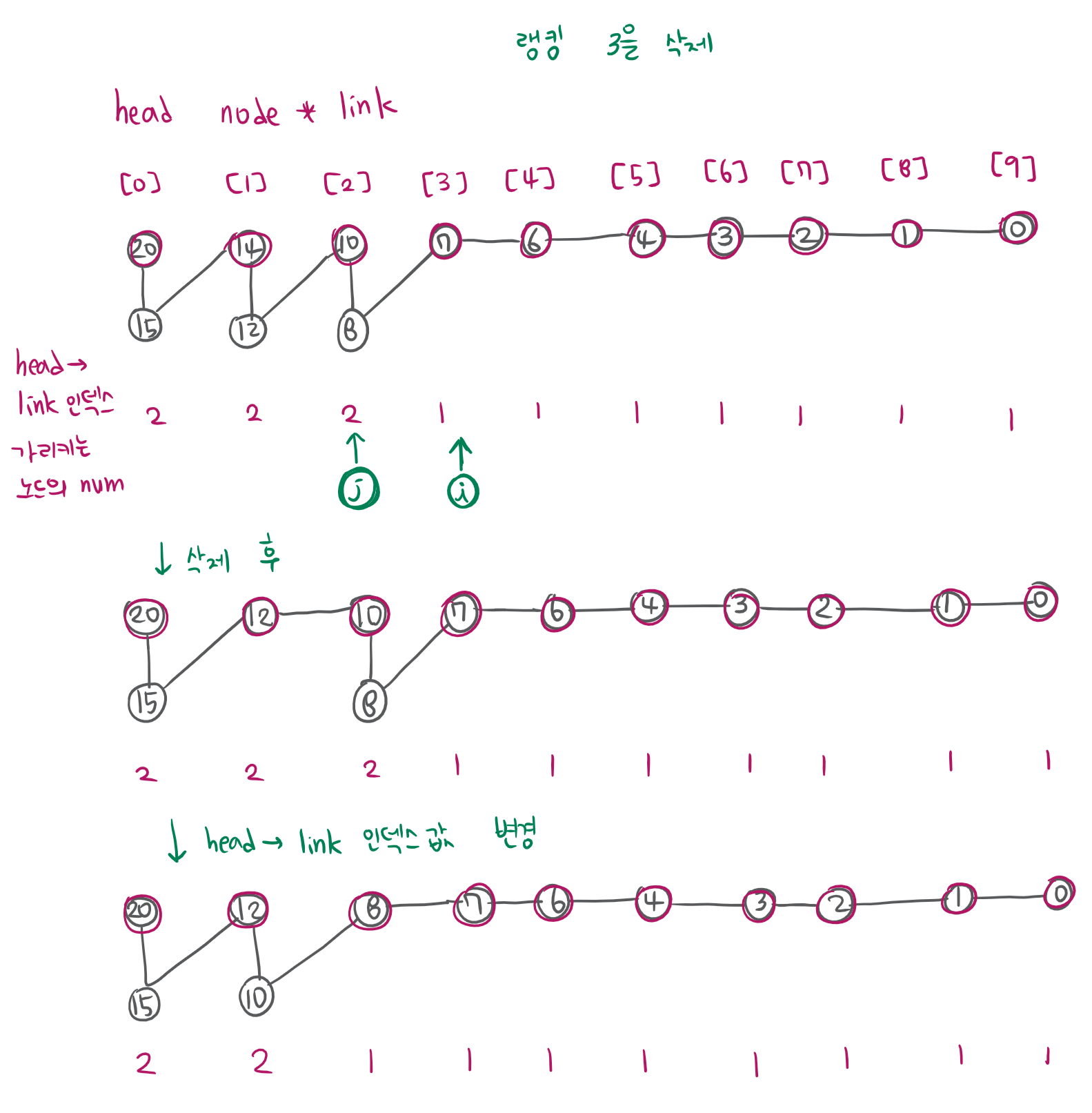
(j-1)이 0이고 (i-1)이 0인 경우와 그렇지 않은 경우 /

(j-1)이 9이고 (i-1)이 9인 경우와 그렇지 않은 경우 )

1. head->link의 인덱스 값 i부터 j-1까지 head->link가 가리키던 노드를 원래 노드의 왼쪽 노드로 변경하고 num값도 변경한다.
2. head->link의 인덱스 값 j부터 i-1까지 head->link가 가리키던 노드를 원래 노드의 오른쪽 노드로 변경하고 num값도 변경한다.

head->link[i-1]의 num을 1감소시키고, 전체 노드 개수도 1 감소시킨다.

예를 들기)



**findbyname():** 입력받은 이름으로 저장된 점수를 출력한다.

head->link[0] ( linked\_list의 맨 앞 노드)부터 살펴보면서 입력받은 이름과 같은 이름을 저장하고 있는 노드를 찾으면 출력한다. 이 시행을 linked-list가 끝날 때까지 한다.

**displayrank()**: 입력받은 x부터 y까지의 랭킹을 보여주는 함수

head->link 인덱스로 접근할 수 있는 노드의 num값에는 자기 자신을 포함한 자기 밑으로 이어진 노드 개수이므로, head->link의 인덱스를 가리키는 i값을 0, sum값을 0부터 시작해서 head->link[i]->num값을 sum에다 더하고 이 sum값이 x보다 작을 때까지 i++하면서 반복수행한다. 이 반복문이 끝나고 i-=1하고 sum이 x보다 커지기 전에 마지막으로 더해준 값을 뺀다. 그러면 i는 x랭킹 노드가 밑으로 들어가 있는 head->link인덱스를 가리키고 sum값은 이런 i-1까지의 num합을 가리킨다. tmp=head->link[i]이라고 하면 tmp는 sum+1 번 째 랭킹 노드를 가리킨다. 그래서 tmp가 x번 째 랭킹을 가리킬 때까지 tmp=tmp->rlink를 수행한다. 그리고 이 노드의 score과 name을 출력하는 수행을 하고 노드를 다음노드로 변경하는 수행을 (y-x+1)번 째 반복한다.

**writeRankFile()**: Linked-List로 저장되어있는 자료구조를 txt파일로 저장하는 함수

head->link[0]이 가리키는 노드부터 노드를 다음 노드로 변경하면서 노드가 NULL을 가리킬 때까지 파일에 출력하고 free시킨다.

1. *recommend를 위한 알고리즘*
2. play 중 추천 위치를 찾아주는 경우

**InitTetris함수 내에서 추가:** 첫 번째 블록을 떨어뜨리는 상황을 저장해놓는 root를 만든다. field를 저장하고 c배열을 초기화한다. 그리고나서 root를 인자로 recommend함수를 호출해서 recommendX , recommendY, recommendR를 저장하게 한다. 그리고 나서 위처럼 DrawBlock함수를 불러서 recommendX,Y,R에 맞게 ‘R’로 블록을 표시한다.

**BlockDown내에서 추가:** 사용자 마음에 따라 블록이 쌓여진 후 새로운 블록이 떨어지기 전에 새로운 블록이 떨어질 추천 위치를 구한다. 먼저, 만들어져 있는 트리를 del\_recnode함수호출로 삭제한다. 그 후, 현재 상태를 저장한 root를 만들고 이를 인자로 한 recommend함수를 호출한다. 함수호출을 통해서 다음에 떨어질 블록에 대한 추천 위치를 recommendX, recommendY, recommendR에 저장하게 한다.

**recommend(RecNode \*root)**

함수 역할: 인자로 들어온 root의 lv값으로 현재 내가 고려하고자 하는 블록이 nextBlock의 lv+1 번째 블록임을 알고, 이 블록을 떨어뜨릴 환경정보가 root->field이고, root->score이다. 블록을 떨어뜨릴 수 있는 모든 위치와 rotate를 고려한다. 모든 경우에 대한 결과값을 각각의 새로 만든 노드의 score과 field에 저장하고 이 노드의 주소값을 root의 c배열에 넣는다. 그리고 이렇게 만든 모든 새로운 노드마다 recommend(새로운 노드) 함수를 호출해 인자로 들어간 새로운 노드의 환경에서 얻을 수 있는 최대점수를 반환받는다. 반환값이 제일 클 때의 rotate, y, x 값을 저장했다가 root의 lv가 0일 때만 이 값을 recommendR, Y, X에 저장한다. 이 말은 진행 중인 게임에서 곧 떨어질 블록이 쌓일 수 있는 모든 경우에 대해 가장 점수를 크게 갖는 경우만을 recommendR, Y, X에 저장한다는 의미이다.

**del\_recnode:**

맨 처음에 root를 인자로 받아서 해당 인자의 child노드로 연결되어 있는 모든 노드에 대해 del\_recnode의 인자로 넣어 재귀 함수로 호출한다. 재귀 함수의 호출은 트리의 leaf node에 도달하면 멈춘다. 재귀함수를 호출 후에 인자로 받은 노드를 free시킨다. 즉, 이 함수는 트리의 postorder 순서로 노드가 지워지게 한닼.

1. 추천 위치를 찾아서 그 위치로 블록을 자동으로 떨어뜨리는 자동플레이 경우

**InitTetris함수 내에서:** menu에서 “ 3. recommended play “를 선택한 경우 전역변수(autoplay)를 설정해서 이 값을 0에서 1로 설정하였다. 그래서 autoplay값을 통해 1이면, InitTetris에서 실습된 작성한 recommend()함수가 아닌 숙제를 위한 구조체를 사용하기 위한 Init\_recommend()함수를 호출하도록 한다.

**BlockDown함수 내에서:** InitTetris 때처럼 autoplay 값을 통해 1이면, init\_recommend함수를 호출하도록 한다. 새로운 블록을 떨어뜨릴 때마다 새롭게 블록을 추천하기 위해서 Init\_recommend()를 호출한다. 또한 이 함수가 호출될 때마다 시간을 표현해준다.

**recommendedPlay함수:**

play함수와 같지만 command를 받아서 Q 또는 q를 입력받을 때 게임을 종료시키고 다른 키를 받을 경우에는 아무것도 수행하지 않도록 한다. 그리고 recommendR과 blockRotate, recommendX와 blockX가 같아지도록 명령을 내린다 (ProcessCommand 사용).

**Init\_recommend함수:**

<현재 field 정보를 root에 저장하는 방법>

\*field에서 최고 높이구하기 – field의 첫번째 행부터 살펴보면서 field에서 차있는 곳을 찾으면 멈춘다. 멈춘 곳이 최고 높이로 root의 max\_h에 저장해둔다.

\*root의 num 배열을 모두 0으로 초기화한다.

\*field를 구한 최고높이 행부터 바닥행까지 제일 왼 쪽 열부터 살펴본다. 하나의 열을 살펴볼 때 1인 값을 가지는 field값를 처음으로 찾으면 이 때의 행이 해당 열의 최고높이에 해당하므로 해당열의 h배열값을 설정한다. 그리고 바닥 행까지 살피면서 field값이 1이면 해당 행의 num 배열값을 ++시킨다. 이러한 수행을 모든 행에 대해서 하면 num 배열과 h배열로 현재 field정보를 저장해둘 수 있다. 그리고 root의 level과 score을 보두 0으로 초기화한다.

그리고나서 modified\_recommend함수를 부른다

**modified\_recommend(recommend\_node \*t)함수:**

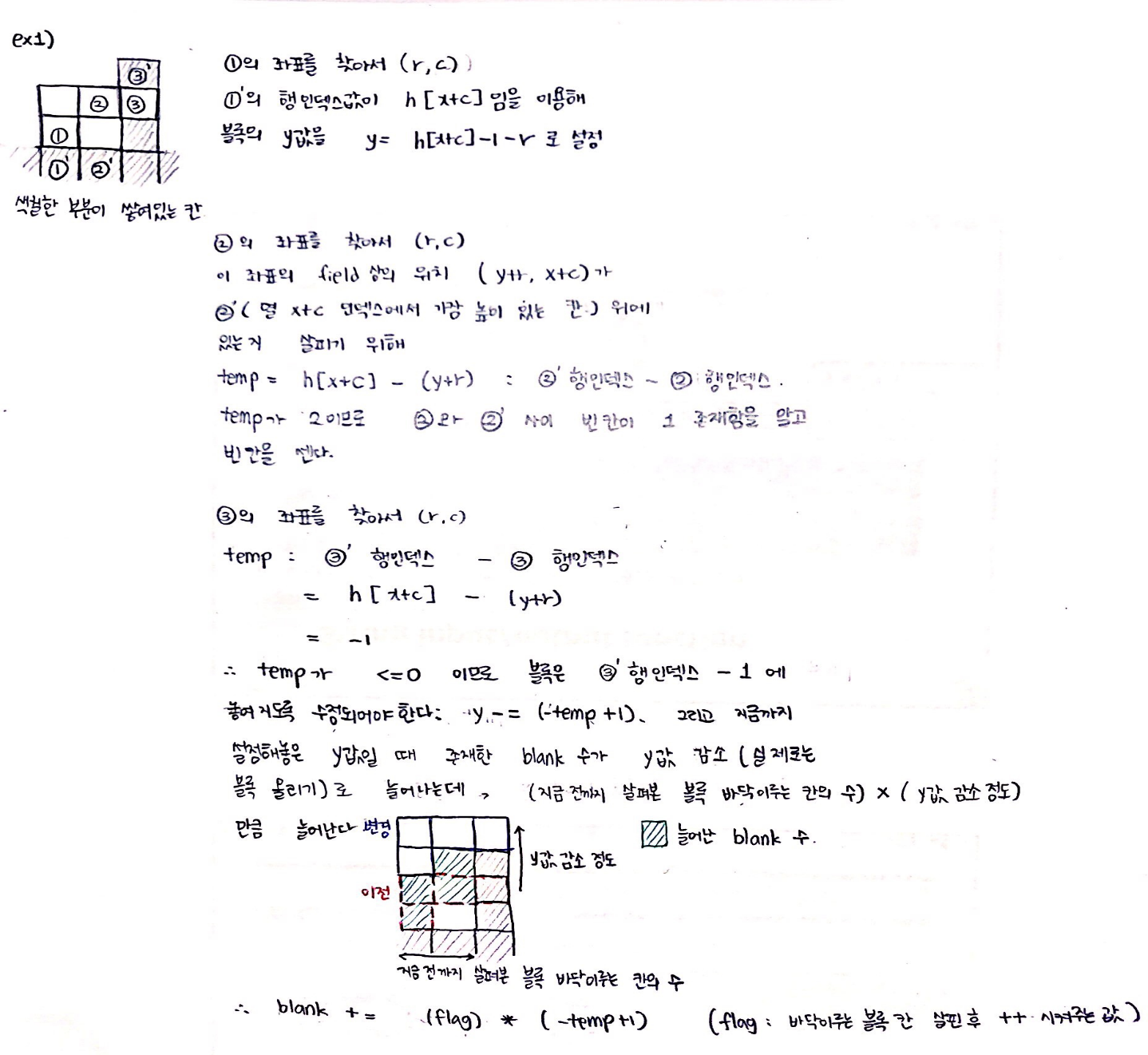
블록을 돌릴 수 있는 경우의 수에 따른 블록을 쌓을 수 있는 모든 x값에 대해 다음을 수행하였다.

1. 블록의 회전수와 x값을 정해놓은 상태에서, 블록의 가장 왼쪽에 있는 것과 오른쪽에 있는 것의 field 상의 위치가 범위를 벗어나지 않는 지 살펴본다.
2. 블록에서 가장 아래에 있는 칸을 찾아서 이 칸이 바닥과 닿도록 블록의 y값을 설정한다.
3. 블록의 아래바닥을 이루는 모든 칸의 좌표를 알아보면서 이미 쌓여있는 블록과 겹치면 블록의 위치를 위로 옮긴다. 즉 y값을 감소시킨다. 그렇게 블록의 아래바닥을 이루는 모든 칸을 살펴보면서 블록이 쌓일 수 있는 y값을 찾는다. 살펴볼 때 지금까지 바닥을 이루는 칸과 필드사이 공백을 세어나가고 바닥과 필드가 맞닿는 면적도 세어나간다. (공백을 세는 이유는 공백이 많이 생기면 이를 바탕으로 파생되는 경우는 고려하지 않고자 판단하기 위한 것이다. 그리고 필드와 맞닿는 면적을 세는 이유는 점수를 계산하기 위함이다.)

블록의 바닥을 이루는 칸을 찾으면, 필드 상에서 같은 열 최고 높이인 행과 지금 칸의 행의 차를 구한다.(temp값)

temp<=0 ) 블록을 놓을 위치가 쌓여있는 블록과 겹치는 경우로 블록이 쌓일 수 있는 위치는 현재보다 한 칸 위에 있어야 한다. 즉, 블록의 y값을 감소시켜서 지금 겹쳐버린 쌓여있는 블록의 바로 위로 블록이 놓이도록 조정한다. 이 때 공백의 수를 조정하고 맞닿는 면적을 1로 초기화한다.

temp = 1 ) 살펴보는 칸이 딱 쌓여있는 블록 바로 위로 놓이는 경우로 맞닿는 면적이 증가하고 y값 조정할 필요없다.

temp > 1 ) 살펴보는 칸이 쌓여있는 블록 위에 붕 떠있는 상태로 y값 조정은 필요하지 않고 공백 수를 증가시켜주어야한다

1. root노드 환경을 바탕으로 쌓아진 블록으로 인한 결과를 저장하기 위한 New\_node를 만든다.

먼저, New\_node의 환경을 root노드 정보대로 초기화해놓은 후, 현재 쌓은 블록으로 인한 결과에 맞게 New\_node의 정보를 갱신한다. 이 때 블록의 모든 칸을 살피면서 아래 과정을 수행한다.

* + 1. 블록을 이루는 칸을 찾으면( 이 칸의 field 내에서의 위치는 행: y+i, 열: x+j ) 이 칸이 위치하는 열의 최고높이를 필요에 따라 갱신해주고(h배열값 변경), 이 칸이 위치하는 행을 채우고 있는 블록 개수를 늘린다.(num배열값 증가)

1. 점수를 계산한다. 이를 위해 지울 수 있는 line을 찾고 h배열과, num배열을 수정해준다. 최고높이 행부터 살펴보면서 바닥까지의 num 배열을 살펴보는 데 한 행을 살펴볼 때마다 아래 과정을 수행한다.
2. num배열값이 WIDTH이면 지울 수 있는 line개수를 증가시키고(line++), 해당 높이의 num배열값은 삭제되고 이보다 높이 있던 num배열값을 한 칸씩 땡긴다.
3. h배열을 살펴보면서 지금 지우는 높이보다 해당 열의 최고높이가 같거나 더 높으면 1만큼 감소시킨다.
4. 그리고 New\_node의 최고 높이를 1만큼 감소시킨다.
5. 지운 line수와 blank값에 따라 만들어진 상황을 이어서 다음을 예상할 지 말지 결정한다. 또한 다음 상황을 모두 예상하여 얻은 최고점수를 가지고 지금까지의 최고점수와 최고점수를 낸 블록정보를 갱신할 지 말 지 판단한다.
6. line을 하나이상 지웠거나 blank가 1보다 작거나 같으면 다음 블록에 대해서도 예상해보기 위해 New\_node를 인자로 modified\_recommend를 다시 호출한다. 이 때의 반환값은 New\_node 상황에서 얻을 수 있는 최고점수가 반환된다.
7. 지금까지의 최고점수보다 더 큰 값이 반환되었다면 최고점수와 블록 정보를 모두 갱신한다.
8. 만약 최고점수는 같은 데 만들어낸 공백 수가 더 적거나 블록이 쌓인 최고 높이가 더 낮으면 이 때의 블록으로 최고점수를 낸 블록 정보를 갱신한다.
9. 지금 떨어질 블록을 고려하고 있던 lv = 1인 상황일 때에만 최고점수를 낸 블록 점수를 recommendX, recommendY, recommendR에 저장한다.

* 이론

Linked List: 각 노드가 데이터와 포인터를 가지고 한 줄로 이어져 있는 방식으로 데이터를 저장하는 자료구조이다. 즉 포인터로 다음 노드의 포인터값을 가지고 있어서 연결성을 알 수 있다.

tree: 서로 다른 두 노드를 잇는 길이 하나뿐이고 loop가 없는 그래프이다. 노드에 데이터와 자신의 밑으로 연결된 노드들의 포인터들을 가지고 있다. 용어로는 최상위 노드인 root노드와 자식 노드가 없는 leaf node, 노드 A가 노드 B의 포인터를 가지고 있으면 A를 B의 부모노드, B를 A의 자식 노드라고 한다.

* 1. 분석
* 프로그램 관련 이론 및 사용법

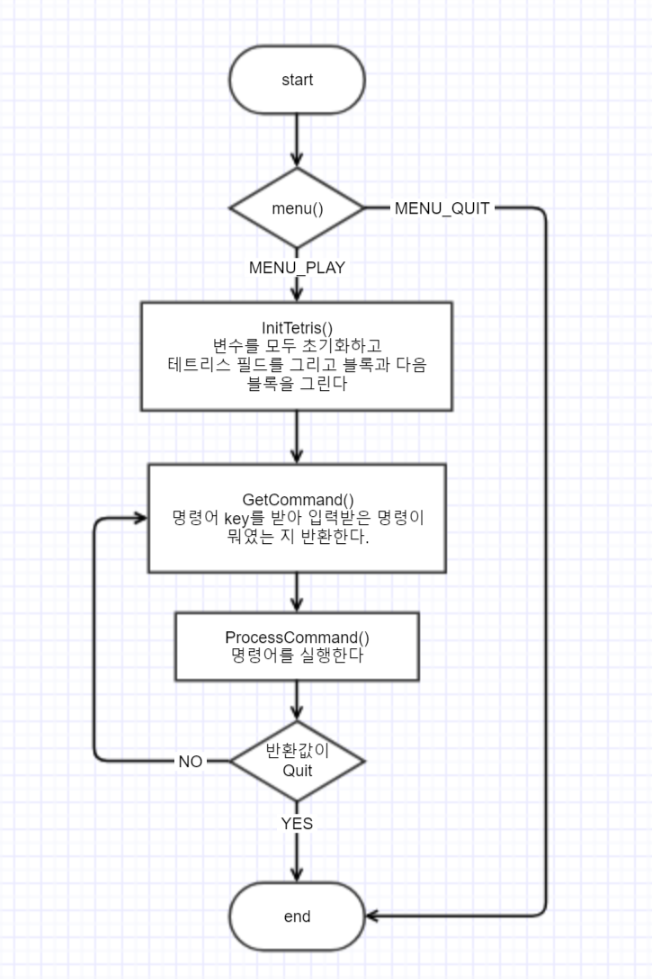
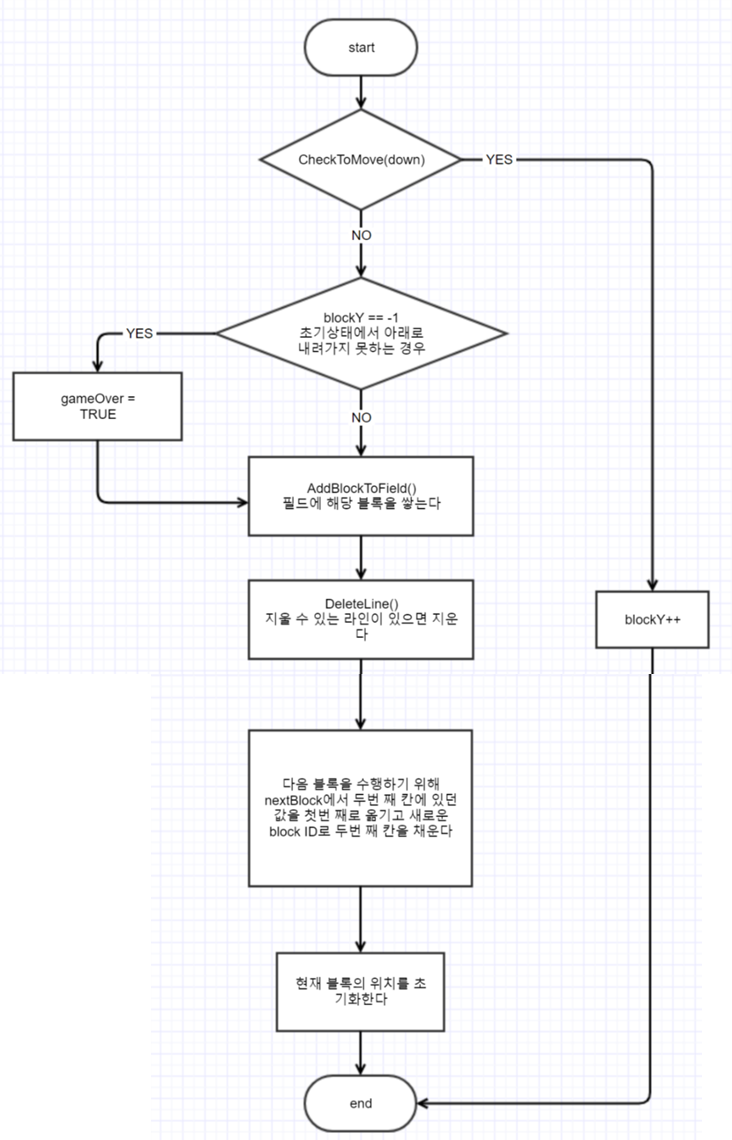
입출력 양식: 평소 쓰던 printf가 아닌 printw을 이용해야 했고 move함수를 통해서 마음대로 커서 위치를 바꿀 수 있다. 또한 출력 후에 refresh()를 쓰는 것이 좋고 echo()함수를 호출하면 입력 글씨가 화면에 나타나고 noecho()로 글씨가 나타나지 않게 효과를 줄 수 있다.

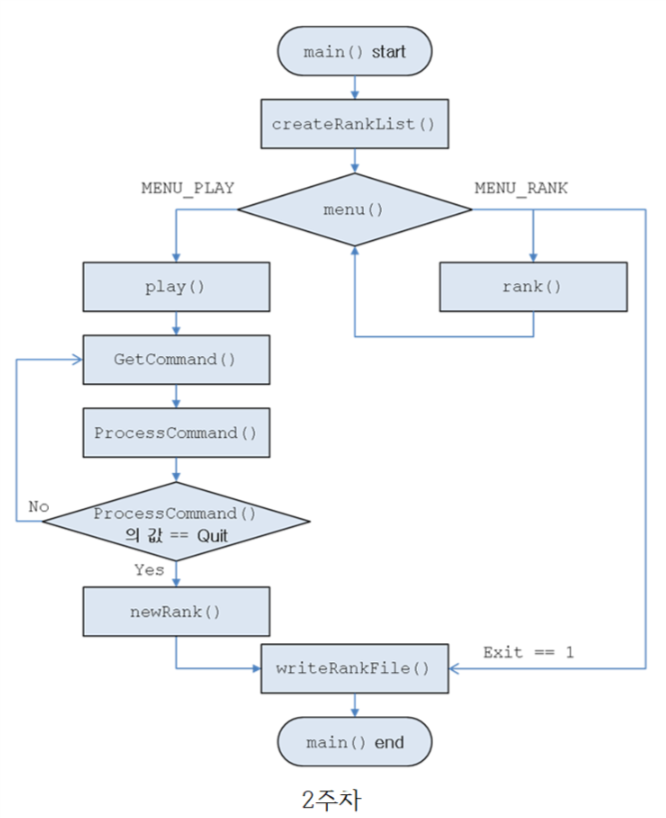
자료구조와 이론: tree 혹은 linked-list 이용할 때 동적할당을 사용하는 데 이 때 적절하게 free를 시켜줘야 프로그램이 터지지 않고 진행할 수 있다.

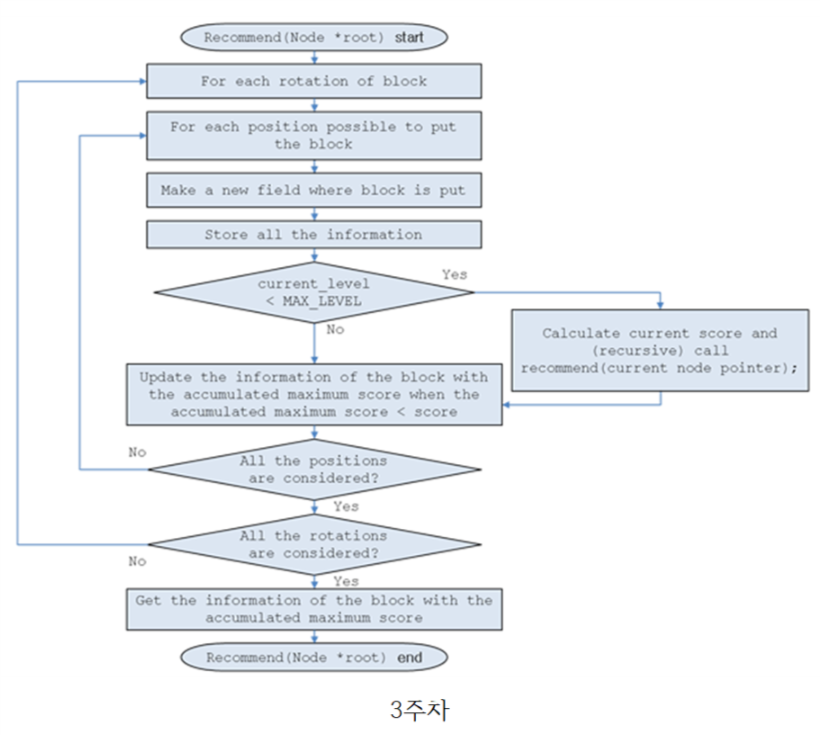
c언어 함수 사용법: strlen으로 문자열길이를 찾을 수 있었고 #define을 적극적으로 활용해서 프로그램 전체 가독성을 높일 수 있다.

* 순서도

**1주차**

play함수 BlockDown함수

 **2주차**

**3주차**

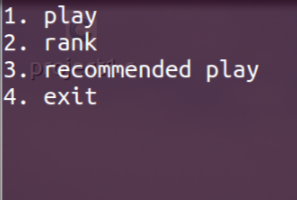
* 1. 제작

1주차: 인덱스 범위를 지정하는 데 꽤나 헷갈렸다. 먼저 CheckToMove함수에서 y가 HEIGHT보다 크면 범위를 벗어나는 것뿐만 아니라 y = HEIGHT 이어도 범위를 벗어남을 알았고, x의 경우에도 x>= WIDTH로 등호가 들어가야됨을 알았다. 또한 DeleteLine에서 바닥행부터 살피면서 한 행을 지우고자 할 때 그 위의 행부터 Max\_Height까지 모두 한 칸씩 아래로 내리는 데, 이러한 작업 후에 다시 현재 행 인덱스부터 살펴야하는 데 다음 행 인덱스를 살피게 해서 오류가 발생했다. 즉 지워야할 두 줄이 연달아 있을 경우 한 줄만 지워졌다. 그래서 지우고난 뒤에는 봤던 인덱스 행을 다시 살피도록 하였다.

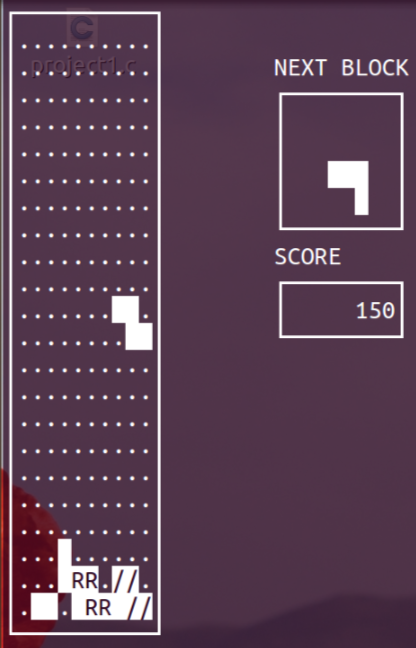
2주차: 내가 구상한 자료구조를 확실하게 사용하기 위해서는 고안해낸 방법이 예외는 없는 지 여러 번 생각해보아야 했다. 여러 번 생각하고 일반화하여 표현한 결과, 생각대로 프로그램이 돌아갈 수 있었다.

3주차: 생각했던 방식으로 한 결과 굉장히 많은 오류를 접했었다. 그래서 디버깅을 해 본 결과 malloc으로 할당해주고 free를 시켜주고 다시 같은 함수에서 malloc을 사용했을 때 이 전에 free 시켰던 노드의 정보가 저장되어 있는 포인터가 반환되었다. 그래서 나의 생각과 다르게 프로그램이 돌아가게 되었었다. 해결방법으로 calloc을 써서 동적할당을 하면서 노드의 데이터가 초기화되도록 하였더니 잘 돌아갔다.

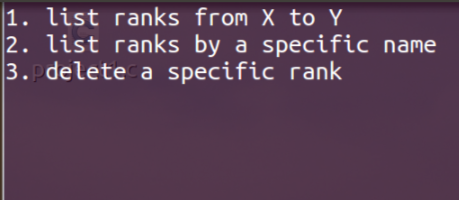
* 1. 시험

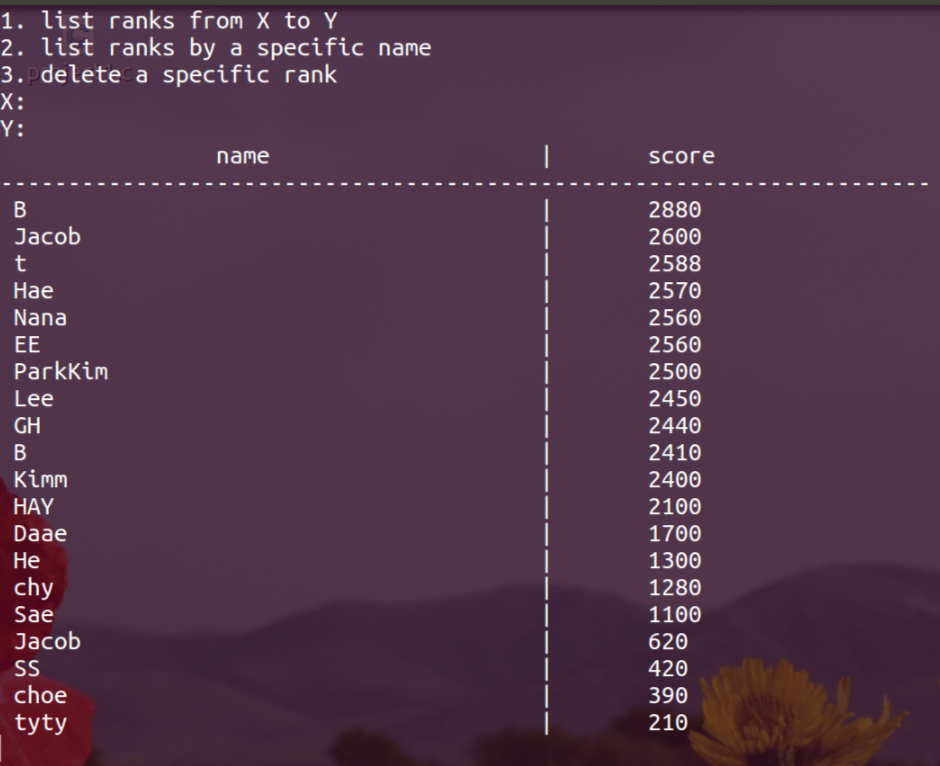


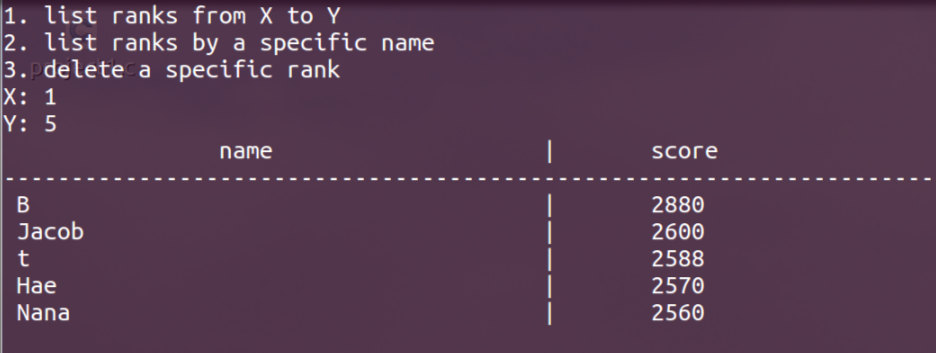
1, play

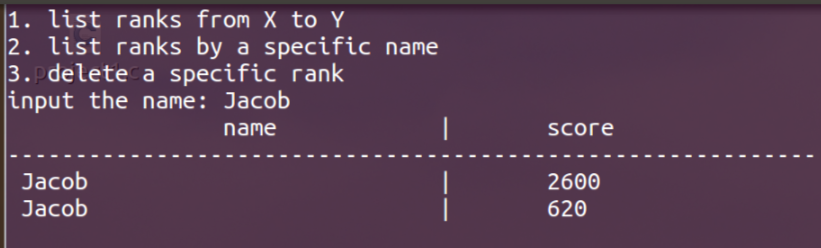
* CheckToMove 함수에 의해 1초마다 한 칸씩 떨어지는 지 확인해야한다.
* 키보드 키 입력에 맞게 블록이 돌아가거나 위치가 바뀌고 또한 필드를 벗어나는 범위로 블록을 바뀌게 하지는 않는 지 살펴보아야한다.
* 블록의 추천위치와 그림자가 잘 표현되는 지 살펴보아야한다
* 한 줄이 다 차면 지워지고 score이 알맞게 올라가는 지 보아야한다.
* 블록 아랫부분과 필드가 만나는 면적에 맞게 점수가 올라가는 지 보아야한다.

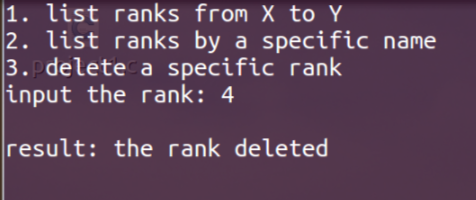
1. rank

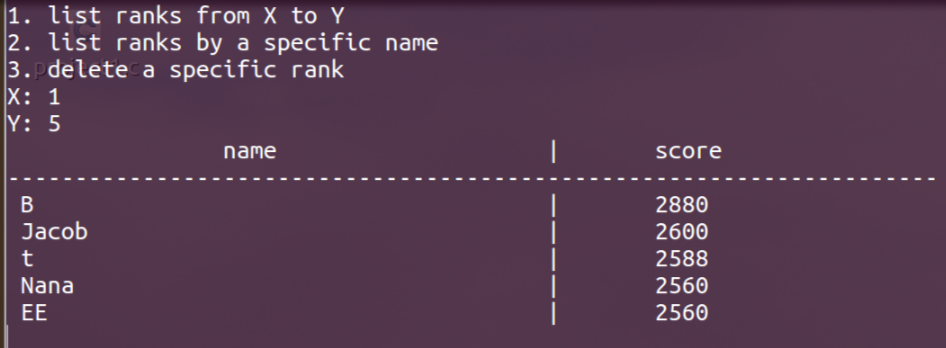




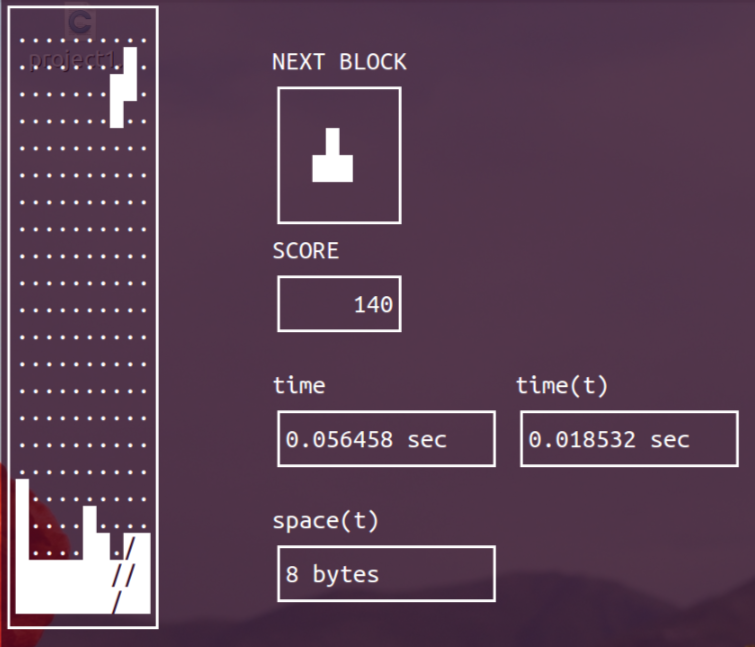








1. recommend



* 1. 평가

게임을 수행할 때 블록이 필드를 벗어나 출력되는 경우가 없었으며 한 줄이 모두 채워지면 바로 지워지고 그에 따라 점수도 알맞게 올라갔다. 또한 게임종료도 알맞게 구현되었다. rank에서도 위에서 볼 수 있듯이 아무것도 입력하지 않으면 전체 랭킹을 보여주고 값을 입력 시 해당 범위 랭킹을 보여주었다. 또한 이름 검색으로 점수를 출력 시 하나가 아니어도 알맞게 출력되었고 존재하지 않는 이름이면 search error 메시지를 띠웠다. 또한 랭킹을 삭제시에도 삭제한 후에 랭킹을 출력하면 그 랭킹이 적절하게 삭제되어 있었다. recommend에서는 play에서 R로 추천위치가 적절하게 표현되었고 recommended play를 위해서는 5분 동안 내가 얻은 점수와 비교해서 평가해보았다.

|  |  |  |
| --- | --- | --- |
| 횟수 | recommend(점) / 5분 | User\_control(점) / 5분 |
| 1 | 730 | 680 |
| 2 | 1090 | 930 |
| 3 | 790 | 660 |
| 평균 | 870 | 757 |

해본 결과 recommend가 내가 얻은 점수보다 더 높은 점수를 얻었다.

* 1. 환경

학생들이 리눅스 서버를 접속하여 프로젝트를 진행하므로 해당 서버에 접속할 수 있는 데스크탑과 ssh 접속 프로그램을 제공한다. 접속하는 리눅스 서버에 각 학생들에게 하위 계정을 발급하여 할당받는 용량에 한하여 자유롭게 이를 이용하여 프로젝트를 진행할 수 있는 환경을 제공한다.

* 1. 미학

1주차: 실습에 추가적으로 그림자 기능으로 블록이 쌓일 위치를 ‘/’으로 표시하는 기능을 추가하였고 BlockNum을 바꾸어도 오류없이 돌아가였다.

2주차: 1주차에 추가적으로 게임이 gameover로 종료되면 이름을 받아 랭킹시스템에 추가시키는 기능을 추가하였다. 또한 1주차와는 독립적으로 랭킹을 출력하고 탐색 및 삭제가 잘 돌아가도록 하였다.

3주차: 1주차에 추가적으로 추천위치가 나타나도록 설정하였다. 그리고 recommended play에서는 지금까지와는 독립적으로 추천위치를 찾고 이 해당 위치로 컴퓨터가 알아서 블록을 옮기는 작동을 하도록 하였다.

* 1. 보건 및 안정

랭킹 시스템 자료구조에서 search라는 사용자 지정 head->link의 배열길이를 바꿔도 오작동하지 않는다. 그리고 랭킹시스템에서 없는 이름을 검색하면 search failure 메시지를 출력하도록 하였고, 또한 해당 범위의 랭킹을 보고자 할 때 x가 y보다 크거나 y가 전체 랭킹 수보다 크면 search failure 메시지를 출력했다. 랭킹을 삭제하는 경우에도 입력받은 랭킹이 1보다 작거나 전체 랭킹 수보다 크면 search failure을 출력하였다. 그리고 랭킹시스템에서 이름이 같아도 점수가 같아도 오작동하지 않았다.

추천시스템의 경우 고려하고자하는 블록의 개수가 늘어날 수록 시간이 너무 오래걸려서 오류상황이 발생할 것으로 보인다.

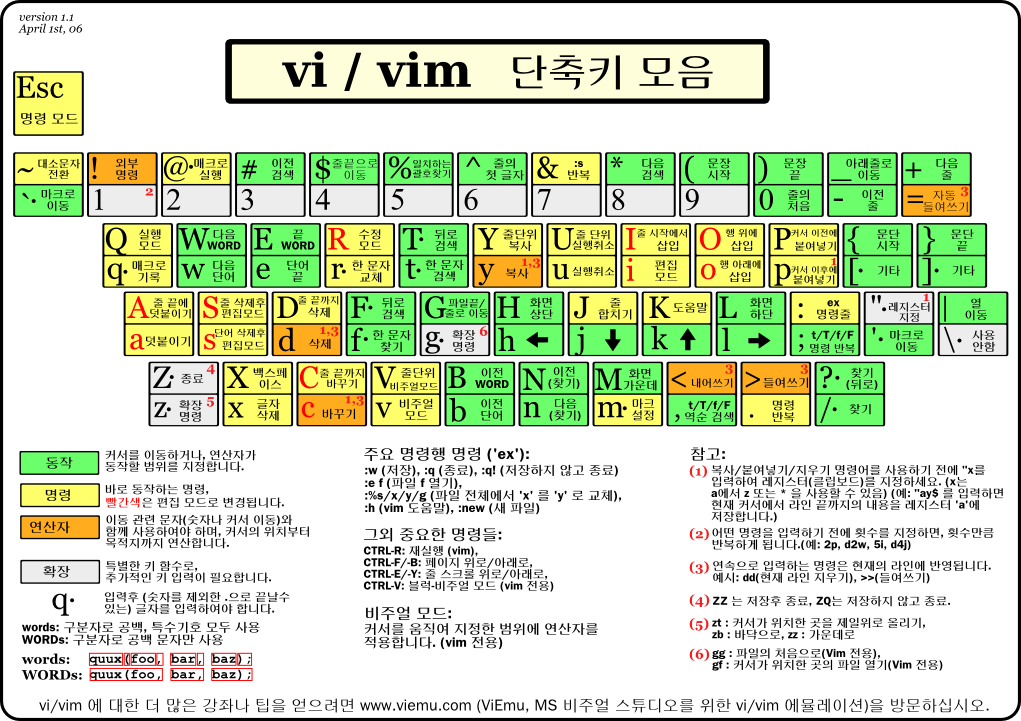
1. **기타**
   1. 환경구성
2. ncurses library

application programming interface(API)가 제공하는 프로그래밍 library이다. 프로그래머들이 terminal-independent한 manner로 text-based user interface를 쓸 수 있게 해준다. 이 library는 스크린에서 커서를 옮기거나 스크린에 나타나는 글씨를 편집하는 등을 간편하게 해주는 함수를 제공한다. 사용하기 시작하기 위해서는 initscr()을 선언해주어야하고 special key (backspace 등)을 받기 위해서는 keypad(stdscr, TRUE)를 선언해주면 된다.

이 라이브러리를 통해서 원하는 크기의 윈도우를 만들 수도 있다. 그리고 윈도우 위에서나 또는 스크린에서 원하는 위치로 커서를 움직일 수도 있다. 그리고 글씨에 색을 입히거나 밑줄을 긋는 등의 속성을 주어서 문자를 출력할 수도 있다. 또한 윈도우의 색도 조정할 수 있고 선을 출력하는 line graphics 변수들도 선언되어 있다.

1. vi

유닉스 환경에서 가장 많이 쓰이는 문서 편집기이다. vi라는 이름은 한 줄씩 편집하는 줄단위 편집기가 아니라 한 화면을 편집하는 비주얼 에디터라는 뜻에서 유래했다. 프로그램을 시작하면 일반적으로 명령 모드로 시작하게 되고 i 키를 누르면 편집모드로 들어가게 된다.



1. gdb

GNU 소프트웨어 시스템을 위한 기본 디버거이다. 여러 프로그래밍 언어를 지원한다. GDB는 컴퓨터 프로그램의 실행을 추적하고 수정할 수 있는 많은 기능들을 제공한다.

gdb를 위해서는 일단 gcc -g -o [프로그램명] [소스파일명] 을 명령해야 한다. 그리고 gdb [프로그램명]을 통해서 gdb가 시작된다. l (몇번째 행)으로 그 행 주변 소스를 볼 수 있고 또는 l (함수이름)으로 함수를 살펴볼 수도 있다. r로 프로그램이 시작되고 b (함수이름)을 하면 함수에서 중단점이 설정된다. 디버깅하면서 s를 누르면 현재 행을 수행하는 데 함수라면 내부로 들어가고 n을 누르면 현재 행을 수행하는 데 함수 경우 함수를 수행하고 나온다. display (변수명) 으로 하나하나의 시행마다 변수 값을 살펴볼 수 있다. 또한 중단점을 info b로 설정한 중단점 정보를 보고 원하는 중단점을 disable로 비활성화할 수 있고 다시 원하면 enable로 활성화할 수도 있다.

* 1. 참고사항

컴퓨터공학 설계 및 실험 1 프로그래밍 강의 자료와 교제

ncurses 위키백과

<http://www.cs.ukzn.ac.za/~hughm/os/notes/ncurses.html#init>

https://ko.wikipedia.org/wiki/Vi

<https://kldp.org/node/102947>

유닉스 리눅스 프로그래밍 필수 유틸리티 : vi, make, gcc, gdb, cvs, rpm

* 1. 팀구성

최민영 개인

* 1. 수행기간

2018.5.7 ~ 2018.5.27