

---

<컴퓨터학 실험 I>

# 테트리스 프로젝트

# 목차

---

- 테트리스란?
- 테트리스 게임의 기본 흐름
- 테트리스 게임의 Flow Chart 및 함수표
- 1주차 구현 내용

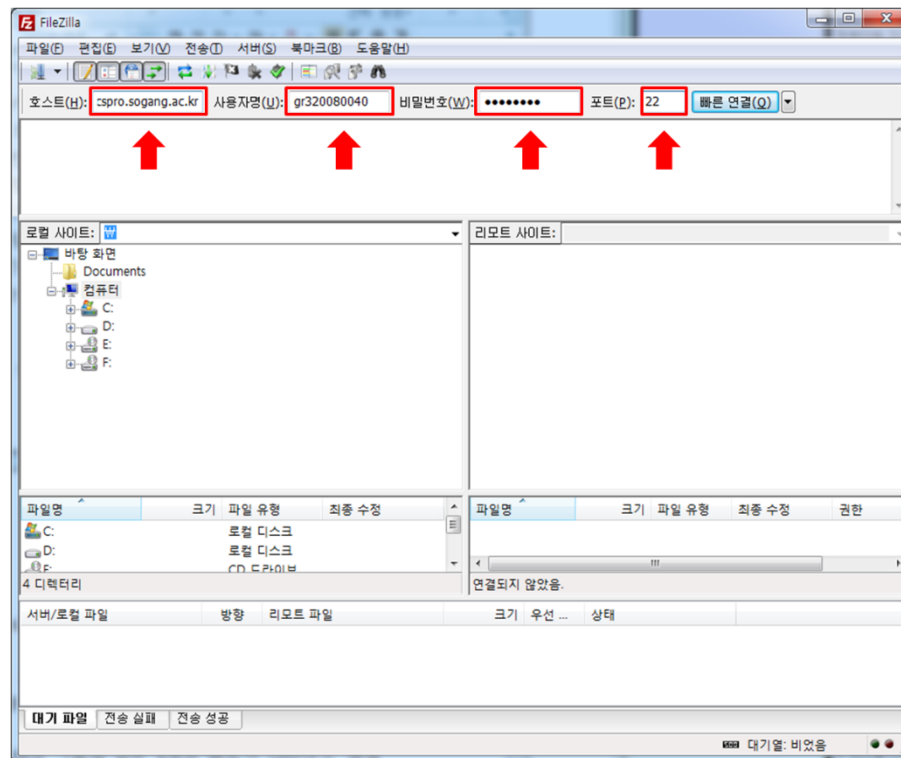
# 테트리스란?

---

- 러시아의 알렉스 파지노프가 개발한 게임이다.
- 직사각형의 빈 공간에 7가지 모양의 블록을 쌓아 빈칸이 없게 되면 사라지면서 점수를 얻는 퍼즐형 게임이다.
- 블록은 한 번에 90도씩 회전할 수 있다.
- 블록은 위 방향을 제외한 방향으로 이동할 수 있다.
- 일정 시간마다 블록이 한 칸씩 떨어진다.

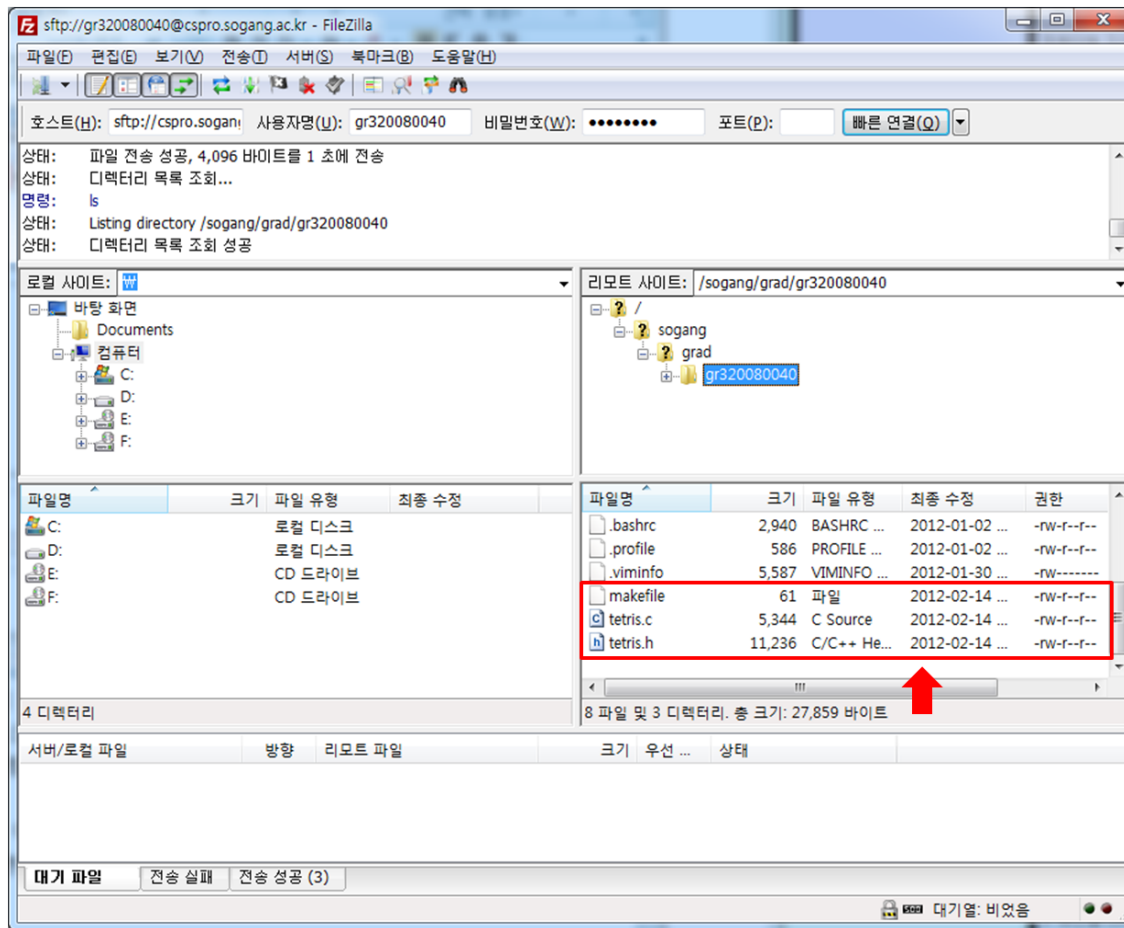
# cspro에 파일업로드(1/2)

- 자신의 계정(cs+학번)에 업로드(upload)하기 위해서 FileZilla 등 파일 전송을 지원하는 프로그램을 다운 받는다. 여기서 FileZilla를 사용한다.
- Sample 프로그램인 a.out을 업로드 해야 한다.
- FileZilla를 설치 후 호스트, 사용자명(cs+학번), 비밀번호, 포트를 입력하고, 빠른 연결을 클릭하여, cspro machine에 접속한다.



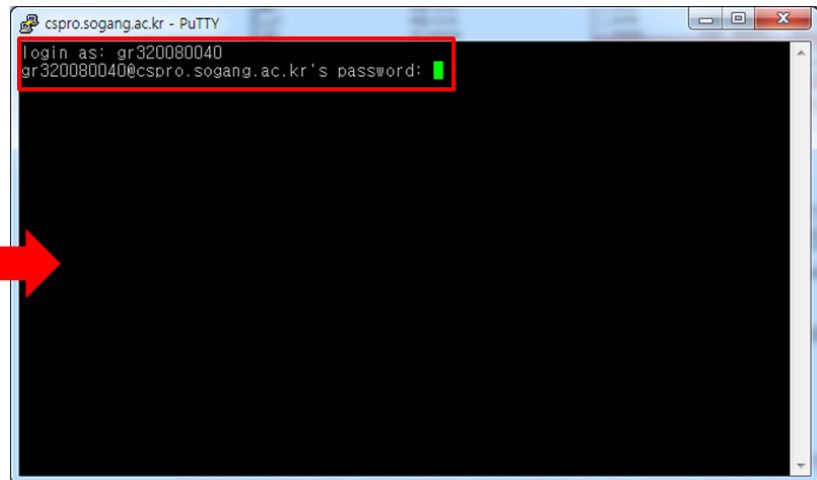
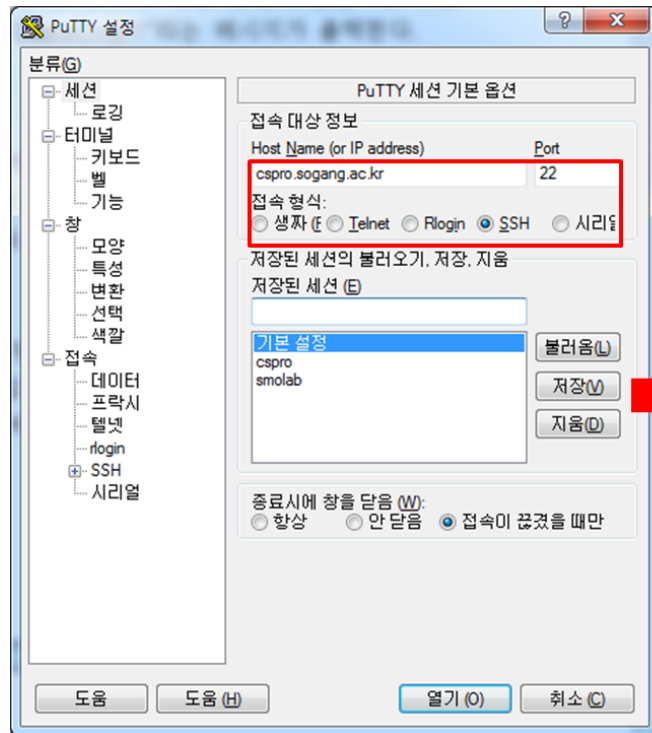
## cspro에 파일업로드(2/2)

- 테트리스 sample 프로그램인 a.out 파일들을 업로드 하기 위해 파일을 선택해서, FileZilla의 오른쪽 아래 창에 드래그 앤 드롭(drag & drop)한다.



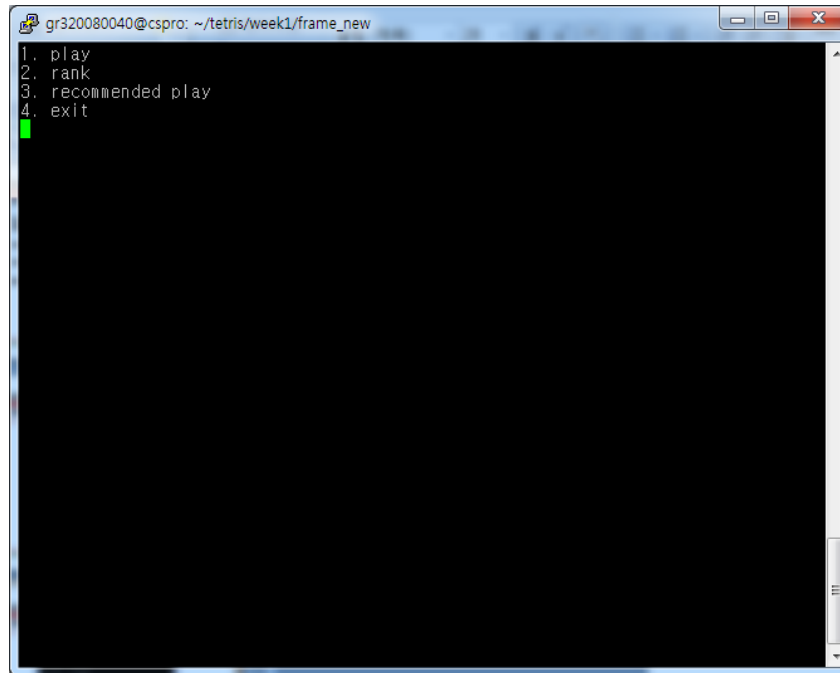
# Sample 프로그램의 실행(1/3)

- ❑ 먼저 cspro server에 접속하여 프로그램을 수행시키기 위해서 “putty 한 글판”을 다운로드하여, 설치한다.
- ❑ Putty를 실행시키고, 호스트 이름 cspro.sogang.ac.kr, 포트 22, 접속형식 SSH로 설정하고 cspro server에 접속한다.
- ❑ 사용자 이름(cs+학번)과 패스워드를 입력하여 자신의 계정에 접속한다.



## Sample 프로그램의 실행(2/3)

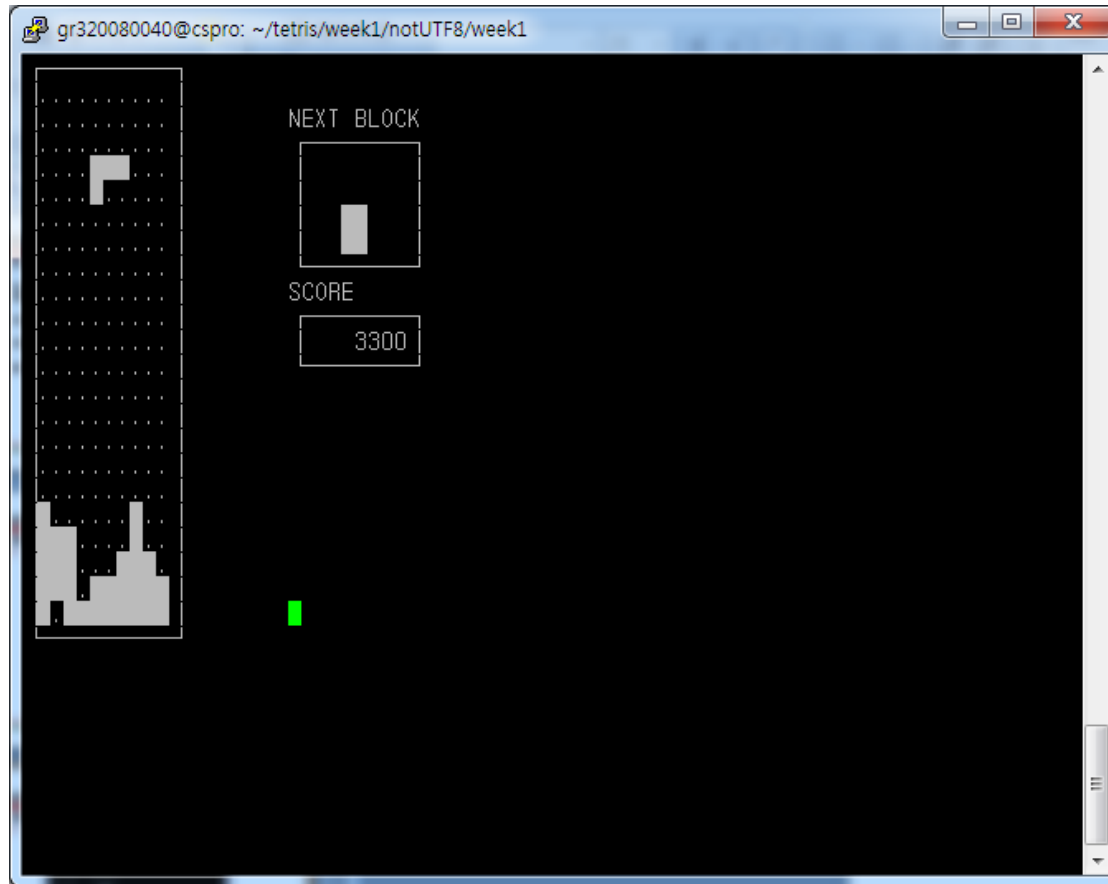
- 업로드된 a.out을 현재 위치의 파일 리스트를 보여주는 “ls”를 입력하여 a.out이 잘 업로드 되었는지 확인한다.
- 터미널에 다음과 같이 명령을 주어 예제 프로그램을 실행한다.
  - ./a.out
  - 처음 실행시키면 다음과 같은 메뉴 화면이 나타난다.
    - 1번 play와 4번 exit 메뉴만 동작한다.



```
gr320080040@cspro: ~/tetris/week1/frame_new
1. play
2. rank
3. recommended play
4. exit
█
```

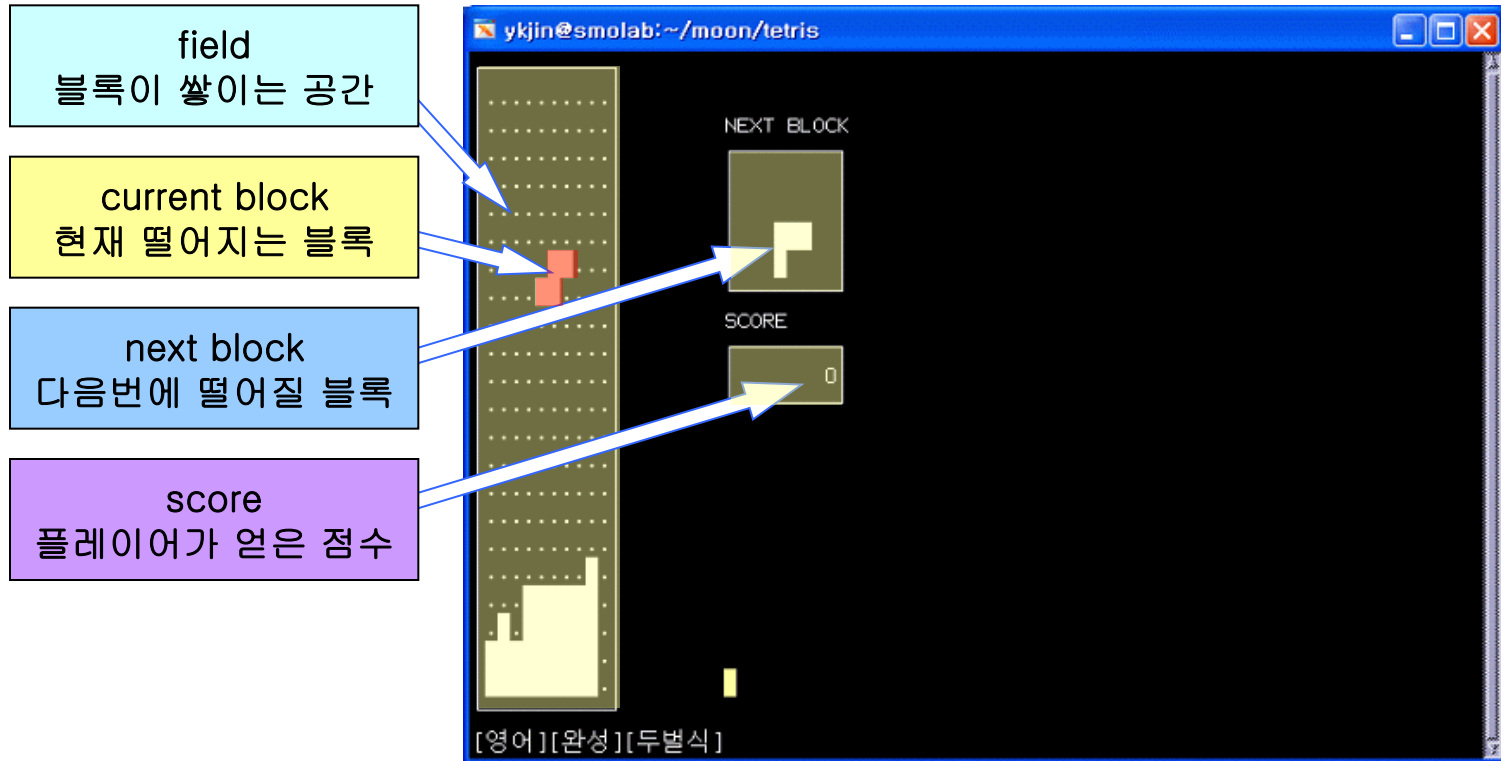
# Sample 프로그램의 실행(3/3)

- 터미널에 다음과 같이 명령을 주어 예제 프로그램을 실행한다.
  - 1번 play 선택하면 다음과 같은 테트리스 게임이 실행된다.





# 테트리스 게임화면 구성




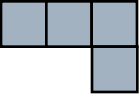
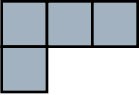
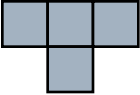

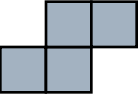
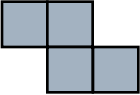
# 필드(field)

- 테트리스의 블록이 쌓이는 공간으로 가로 10, 세로 22의 크기를 가진다.
- 이미 알고 있는 좌표  $(x, y)$ 와는 달리  $(y, x)$  좌표로 필드의 위치를 표현할 수 있고, 기준은 왼쪽 상단이다. 테트리스 필드는 왼쪽 그림과 같다.

	0	1	2	3	4	5	6	7	8	9	x 축
0											
1											
2											
3											
4											
⋮											
⋮											
⋮											
15											
16											
17											
18											
19											
20											
21											

y 축

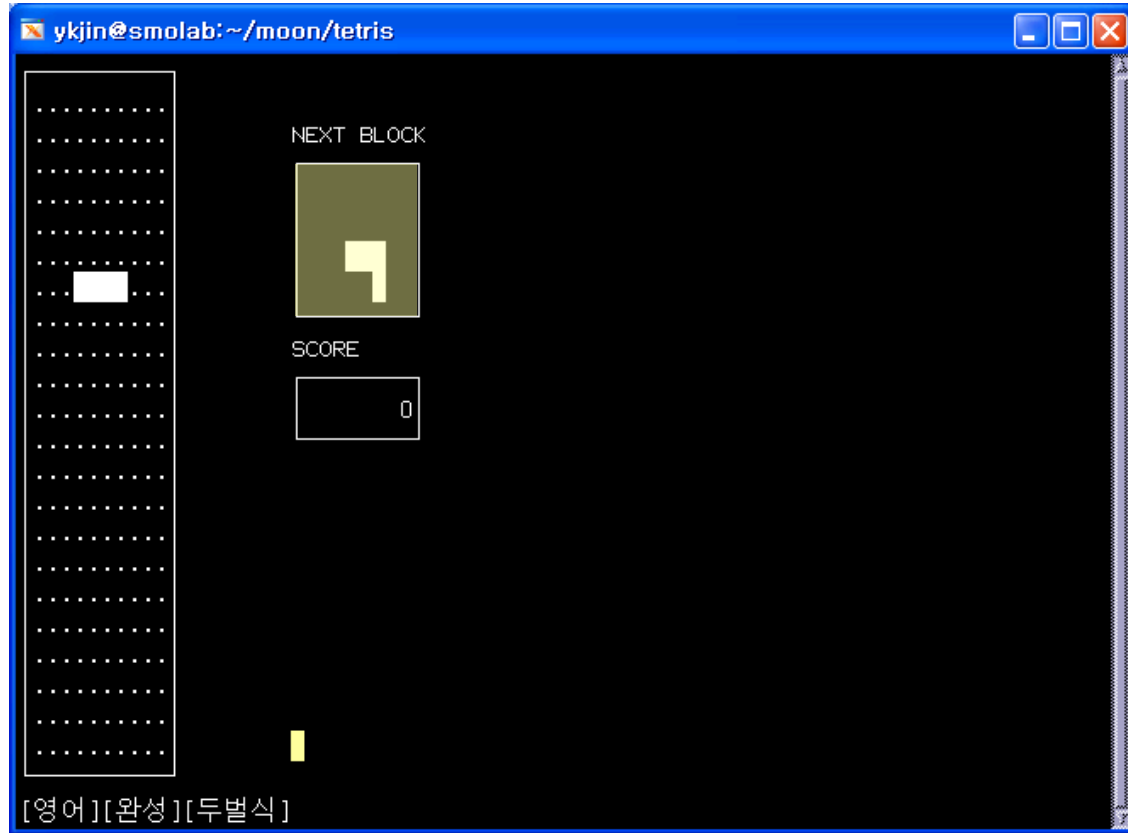
# 블록(block)

shape ID	0	1	2	3	4	5	6
블록의 종류							

블록의 회전은 시계방향과 반시계방향으로 있고  
 각각은 0도에서 90° shape ID로 표현된다.  
 1 : 90° 회전  
 2 : 180° 회전  
 3 : 270° 회전  
 으로 표현할 수 있다.

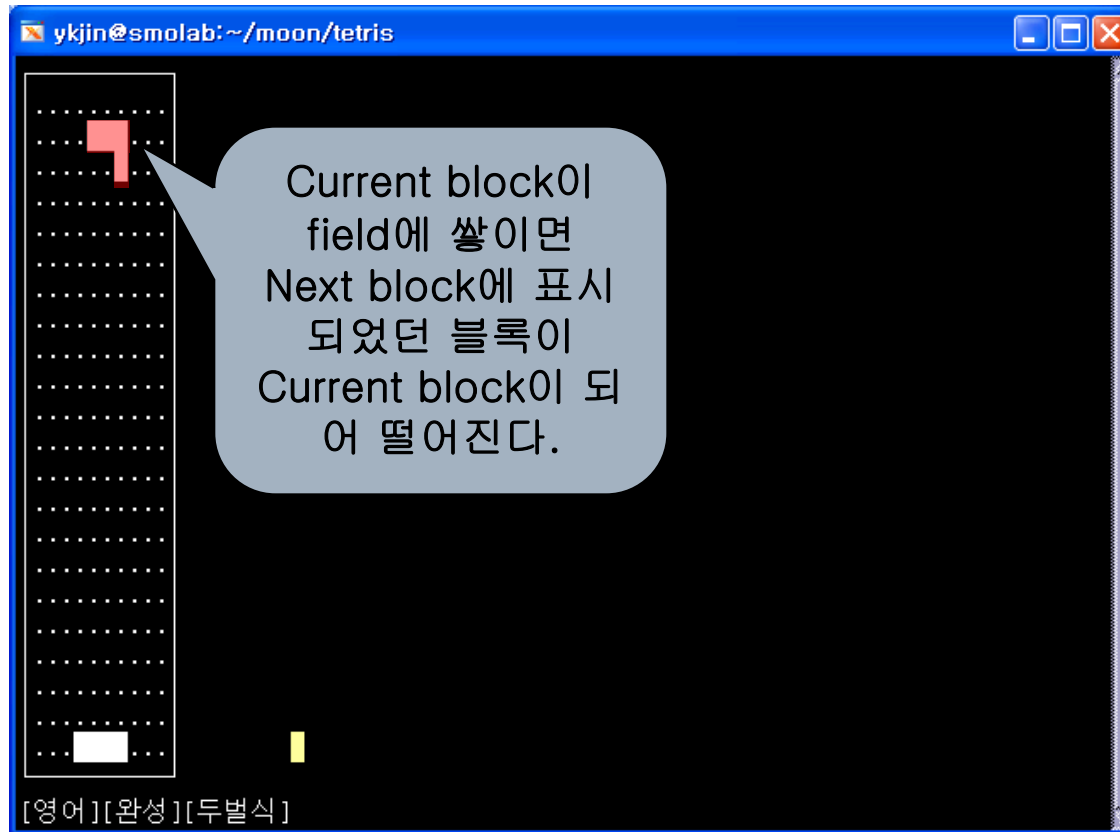
# 게임진행(1/4)

- Next Block은 오른쪽 위 Next Block을 표시하는 상자에 표시된다.



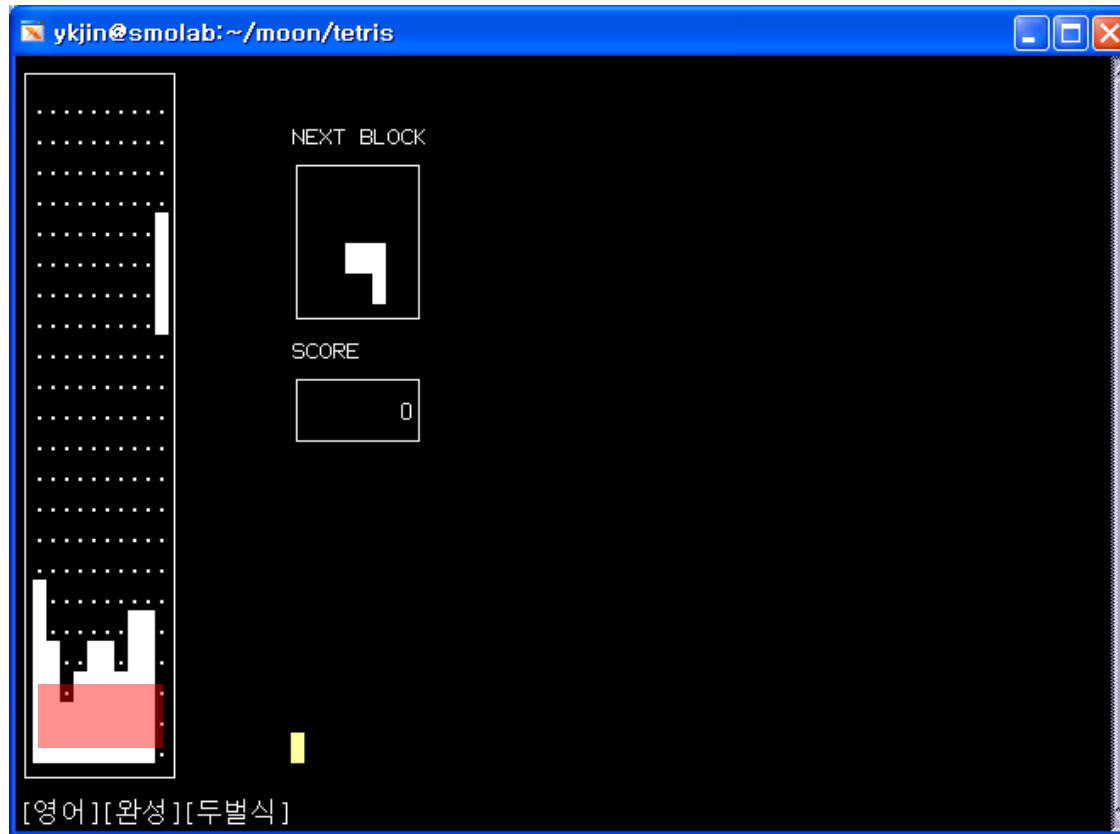
## 게임진행(2/4)

- 현재 블록이 더 이상 아래로 내려갈 수 없을 때, 필드에 쌓이고, Next Block에 있던 블록이 필드에 나타난다.



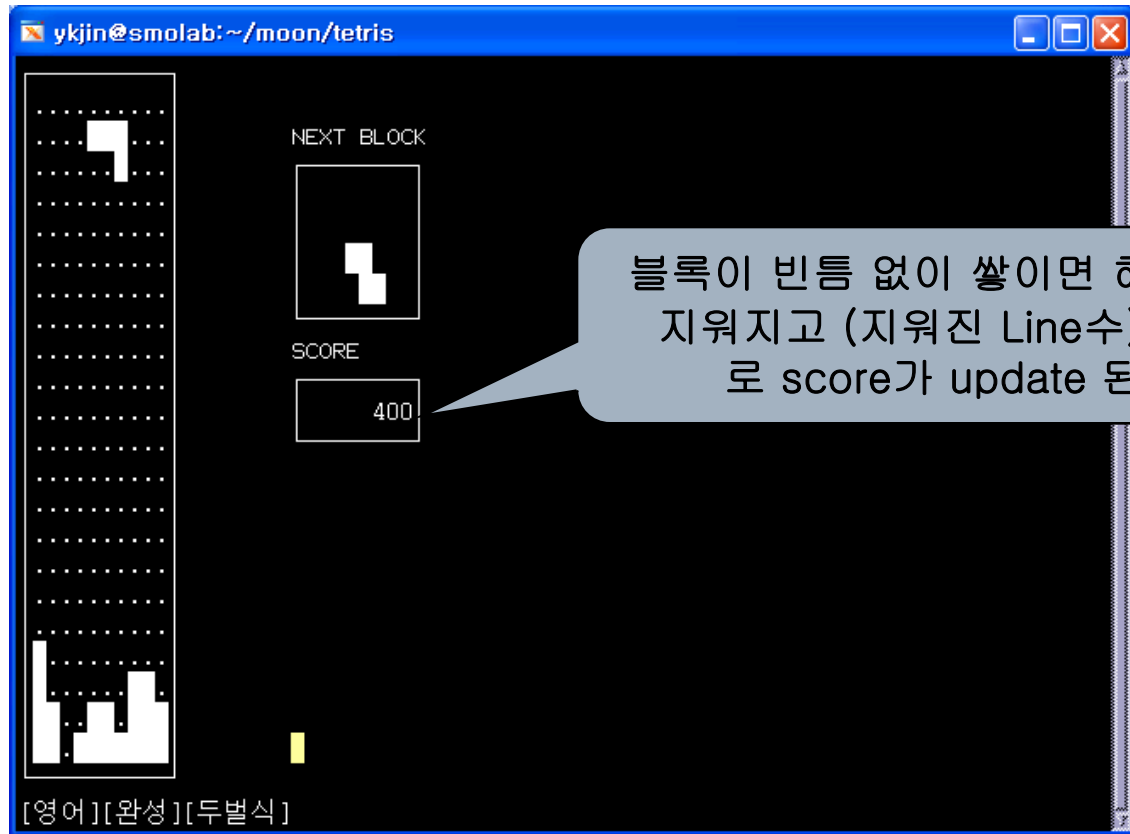
# 게임진행(3/4) – Score

- 필드에 블록이 쌓여, 빈칸이 없는 줄이 생기면 그 줄은 지워진다.



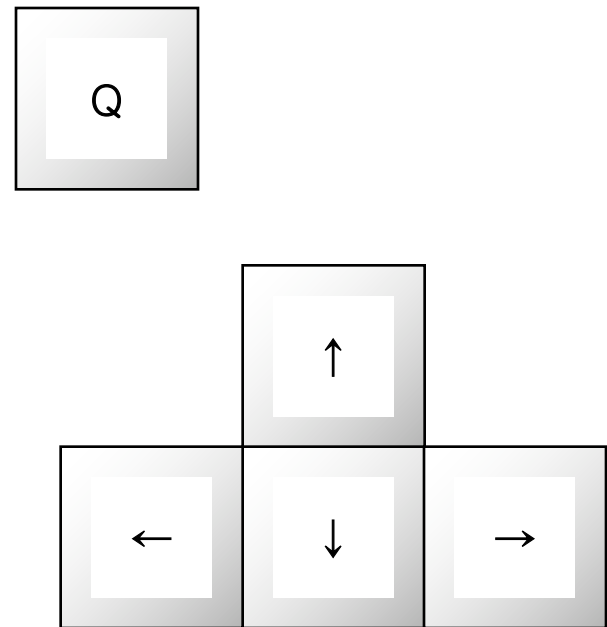
# 게임진행(4/4) – Score

- 줄이 지워지면, 지워진 줄의 수에 따라 score가 계산되고 증가한다.



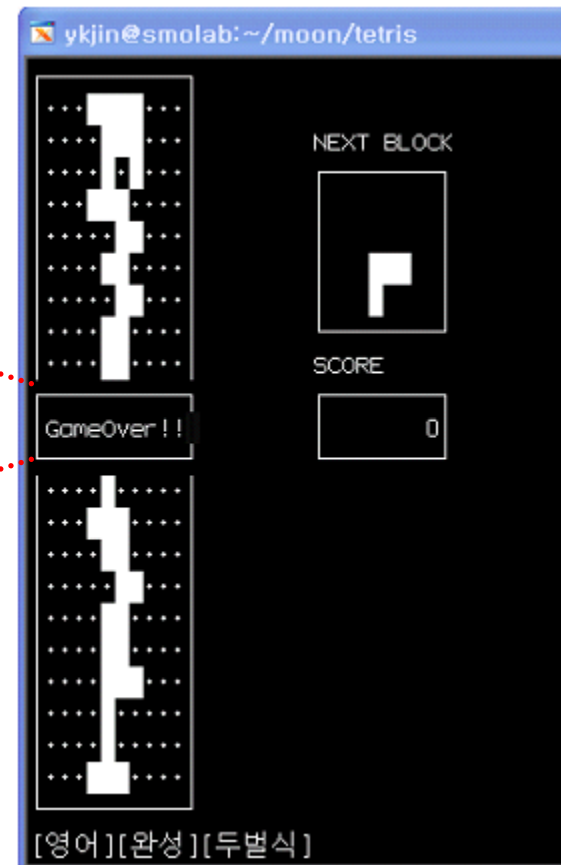
# 키에 따른 동작

---





# 게임이 종료되었을 때



---

## 테트리스 게임의 기본 흐름

Function() : 구현할 함수

Function() : 제공되는 함수

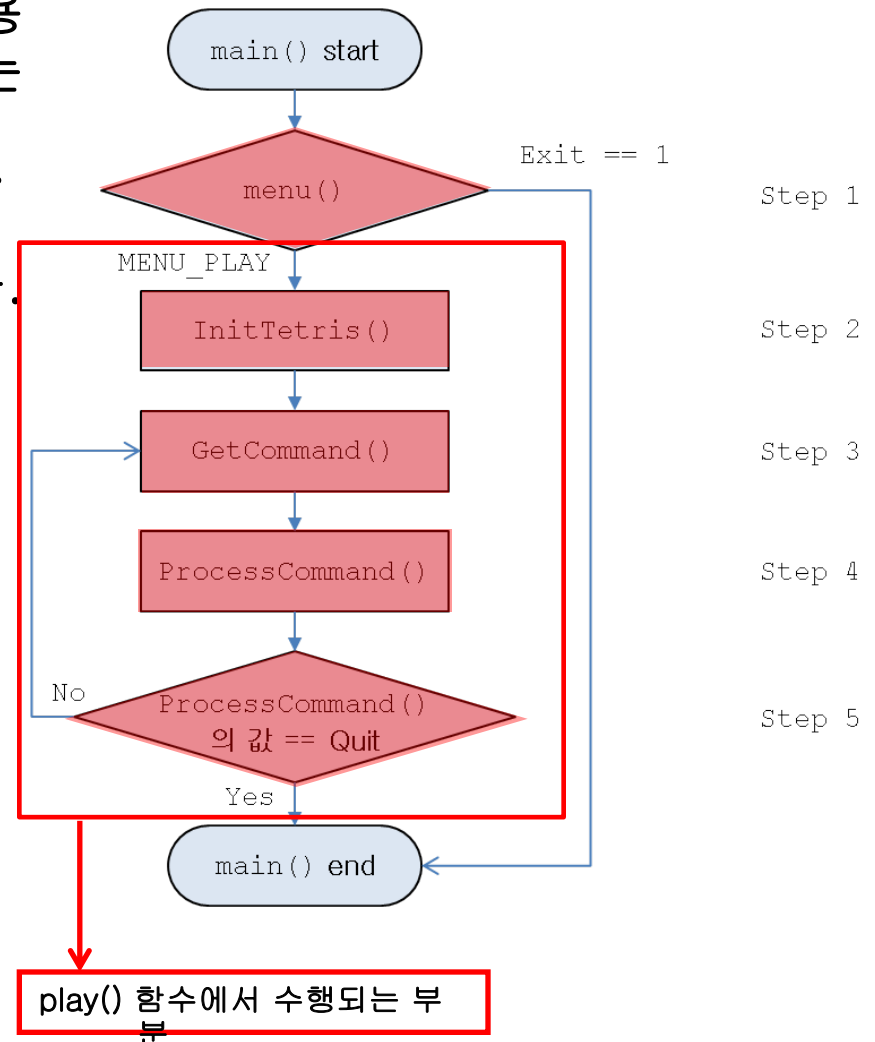
# 테트리스 게임의 기본 흐름

---

- 테트리스 게임은 다음과 같이 두 가지 독립된 처리 과정으로 구성된다.
  - 사용자 키 입력에 대한 동작(`play()` 함수)
    - 사용자로부터 키 입력을 받아 블록을 이동시키거나 회전시키는 일을 한다.
    - 키 입력에 대한 동작은 크게 입력 대기, 입력 처리 두 가지 동작의 loop로 볼 수 있다.
  - 매 초(혹은 지정된 시간)마다 자동으로 수행되는 동작(`BlockDown()` 함수)
    - 블록을 매 초(혹은 지정된 시간)마다 한 칸씩 아래로 떨어뜨린다.
    - 테트리스 필드에서 블록이 빈 칸 없이 꽉 차면 그 줄을 삭제하고, score를 갱신한다.

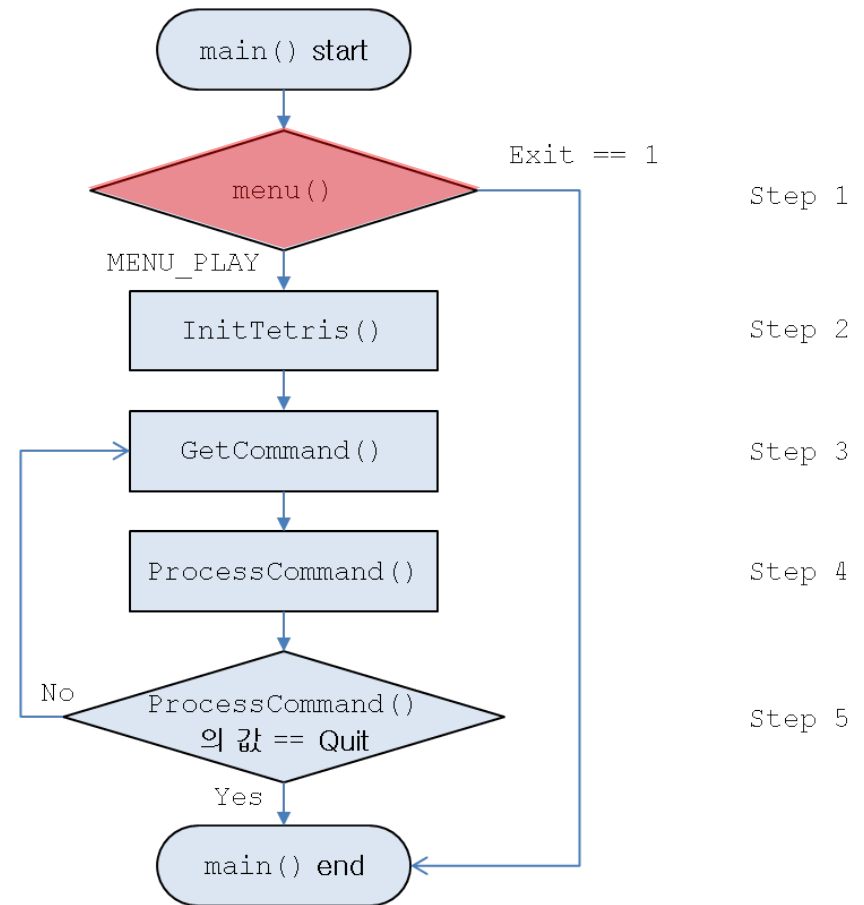
# 키 입력에 대한 세부 동작(main() 함수)

- 테트리스 메뉴를 출력하고, 사용자로부터 메뉴 번호를 입력 받는다.
- 테트리스의 정보를 초기화한다.
- 사용자로부터 키 입력을 받는다. (입력대기)
- 키 입력이 있을 경우, 키 입력에 따라 블록을 이동하거나 회전할 수 있는지 확인한 후, 명령을 수행한다. (입력처리)
- 종료키(Q)를 입력하였으면 프로그램을 종료하고, 아니면 Step 2로 가서 다음의 과정을 반복한다.



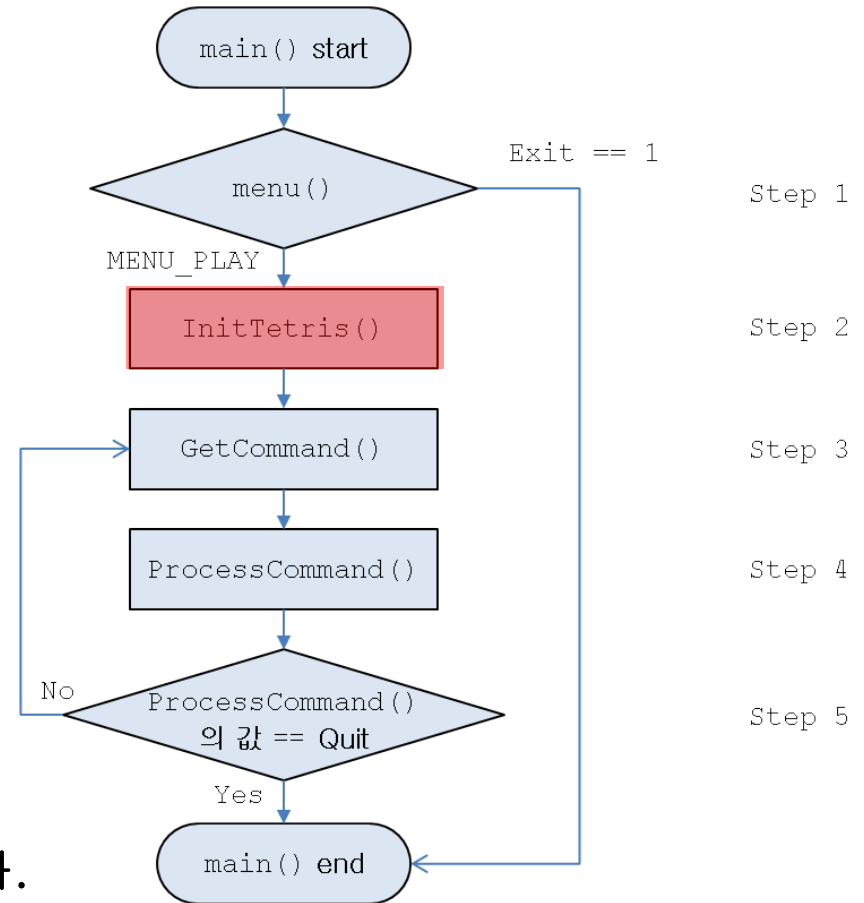
# 키 입력에 대한 세부 동작(menu())

- ❑ 화면에 메뉴를 출력하고 사용자로 부터 입력을 받아서 어떤 메뉴를 수행할지 구분한다.
- ❑ 메뉴 구성
  - 1. play(1주차 실습 및 숙제, 3주차 실습)
  - 2. rank(2주차 실습 및 숙제)
  - 3. recommended play(3주차 숙제)
  - 4. exit
    - 1주차: 기본 테트리스 게임을 구현.
    - 2주차: 랭킹을 등록할 수 있는 랭킹 시스템을 구현.
    - 3주차: 블록이 놓을 위치를 사용자에게 추천하는 추천 시스템을 구현.



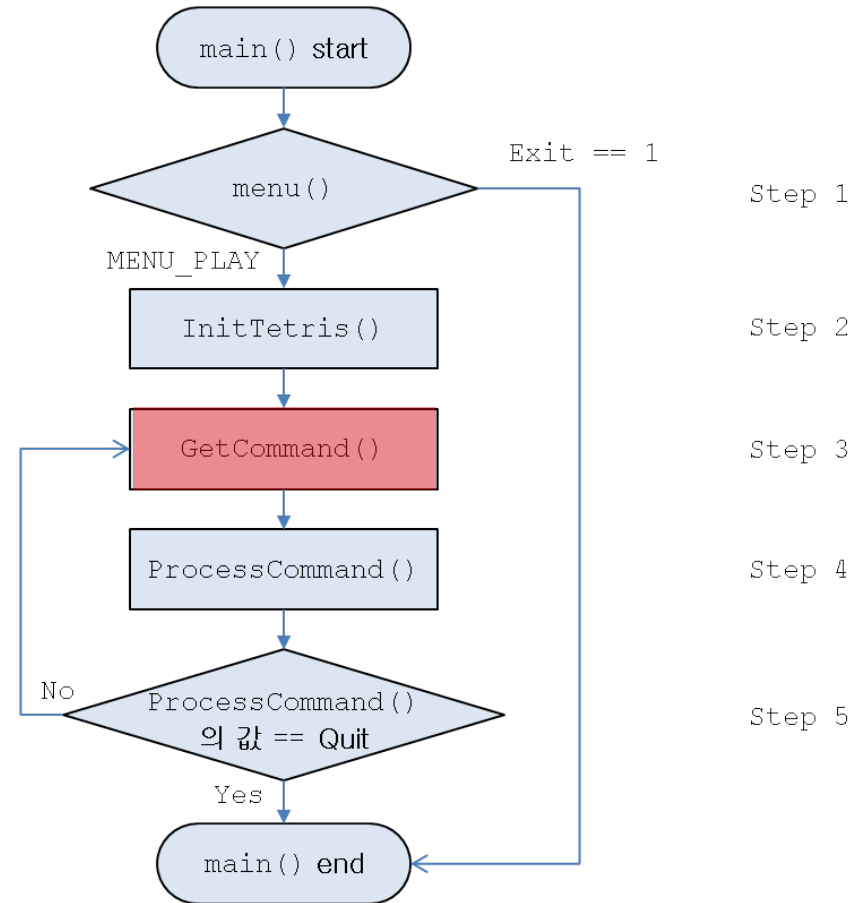
# 키 입력에 대한 세부 동작(`InitTetris()`)

- global 변수를 초기화 한다.
- 테트리스의 기본 틀을 그려준다(한 번 그리면 바뀌지 않음).
  - `DrawOutline()`
- Next block을 화면에 그려준다.
  - `DrawNextBlock()`
- Field를 화면에 그려준다.
  - `DrawField()`
- Score를 화면에 출력한다.
  - `PrintScore()`
- Current block을 field상에 그려준다.
  - `DrawBlock()`



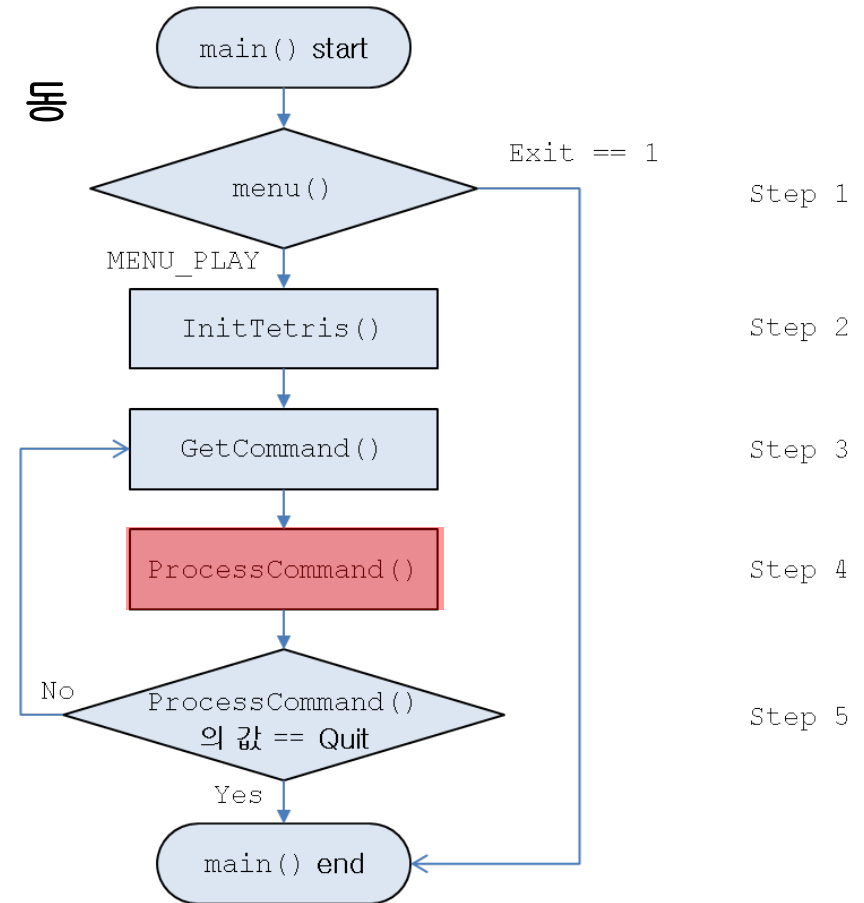
# 키 입력에 대한 세부 동작(GetCommand())

- 사용자의 키 입력이 있을 때까지 대기한다.
- 사용자의 키 입력에 따라 지정된 command를 return한다.
- 프로그램이 처리하는 사용자 입력은 다음과 같다.
  - 상/하/좌/우 방향키 - 블록을 이동하거나 회전한다.
  - 'q/Q' 키 - 프로그램을 종료한다.



# 키 입력에 대한 세부 동작(ProcessCommand())

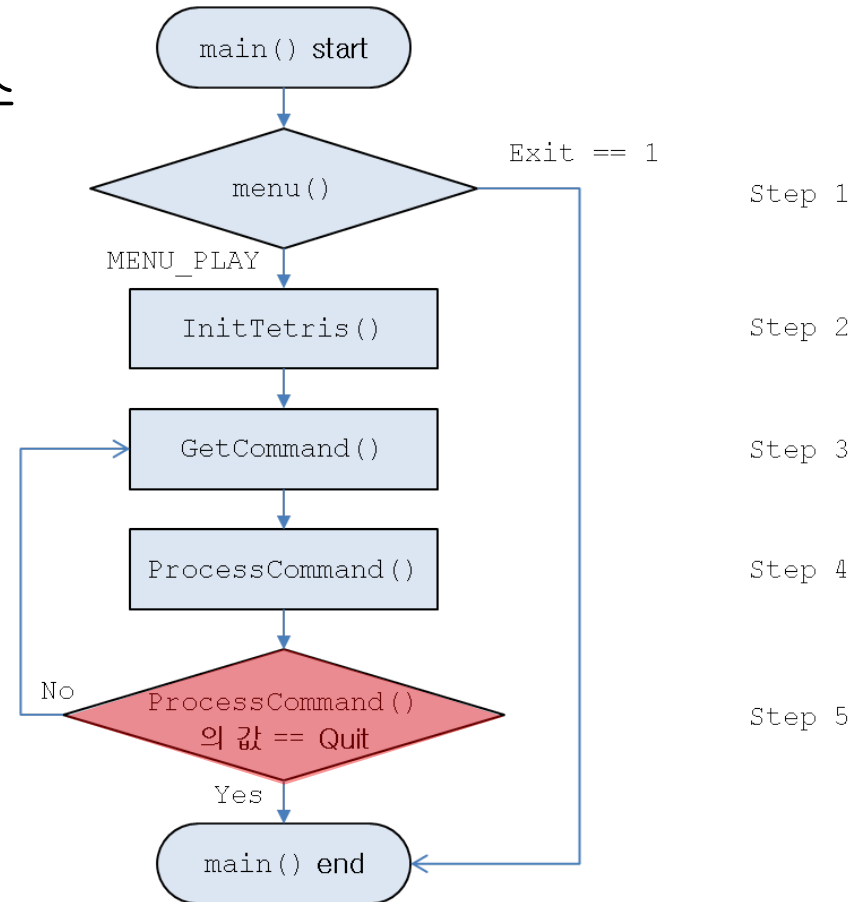
- GetCommand()로부터 받은 command에 대한 current block의 동작이 가능한지를 check한다.
  - CheckToMove()
- 가능하다면 command를 수행한다.
  - DrawChange()





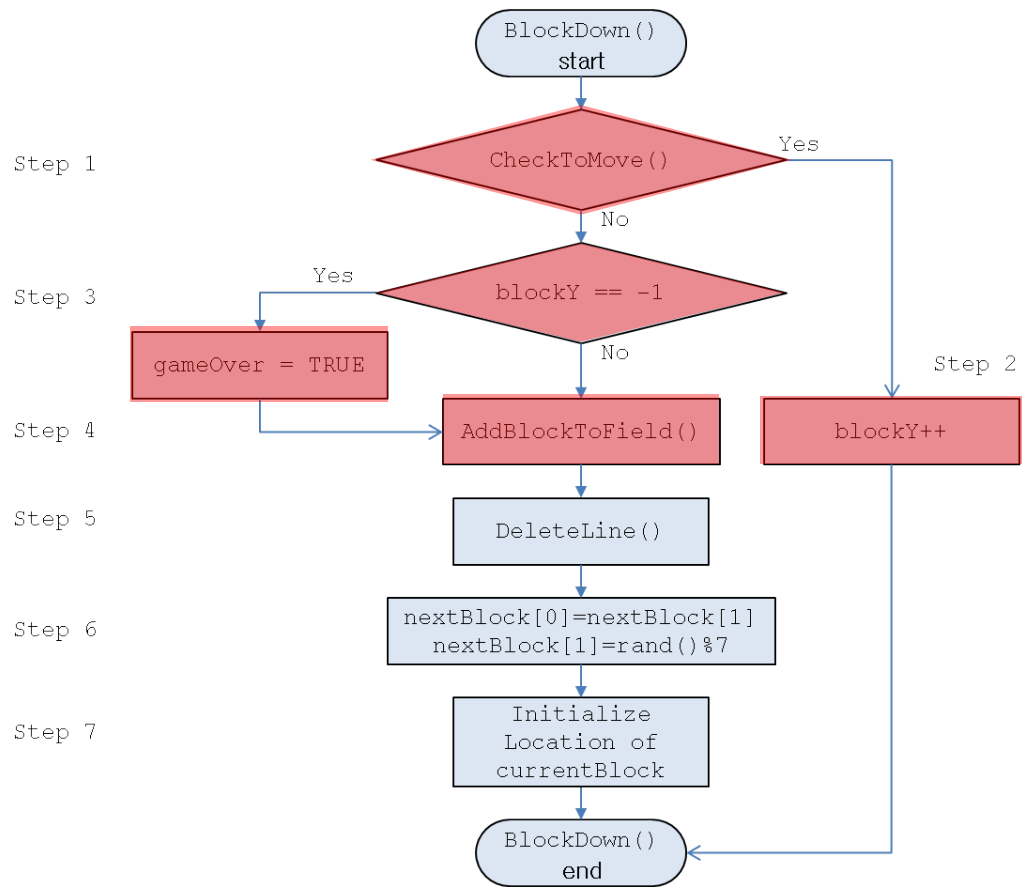
# 키 입력에 대한 세부 동작(종료 조건 check)

- `GetCommand()`로부터 받은 `command`가 `QUIT`일 경우 테트리스 프로그램을 종료한다.
- 아닐 경우 다시 step 2를 수행한다.



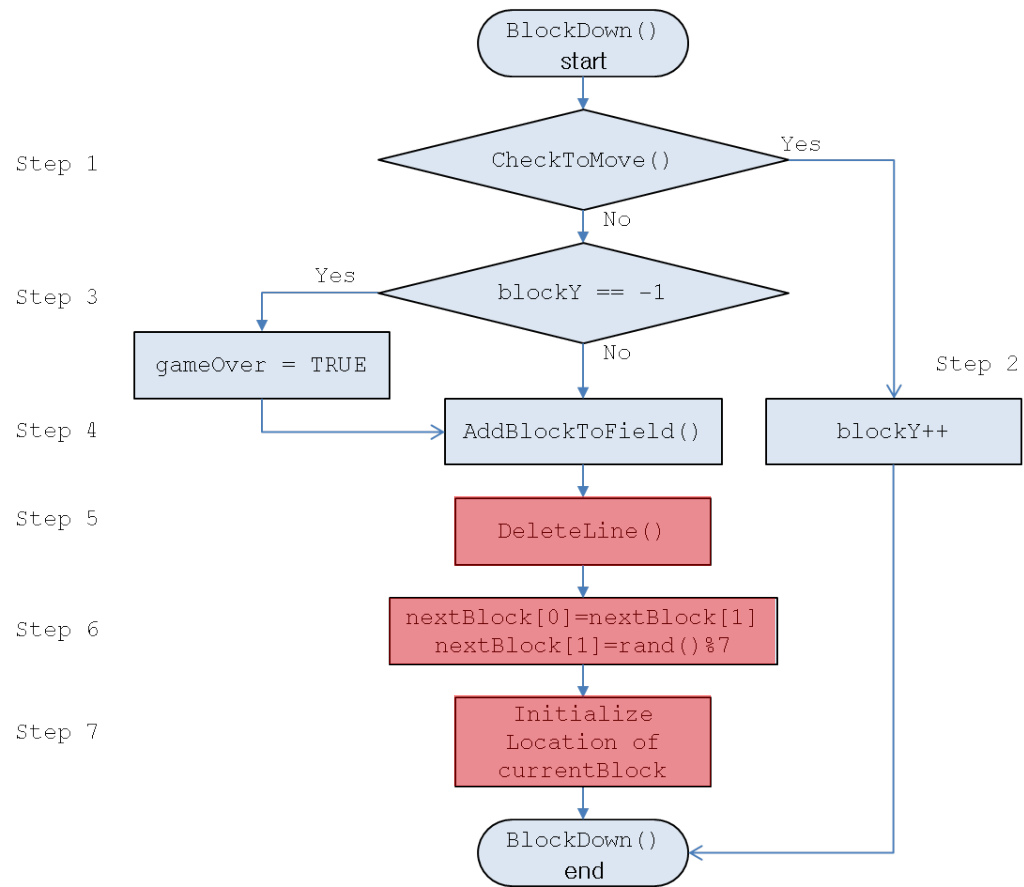
# 1초마다 수행되는 동작(BlockDown()) 함수)(1/2)

- 블록이 한 칸 내려갈 수 있는지 확인한다.
  - **CheckToMove()**
- 내려갈 수 있으면 블록을 아래로 한 칸 내리고 함수를 종료하고, 내려갈 수 없으면 다음 step을 수행한다.
  - **DrawChange()**
- 블록의 y좌표가 초기값인 -1인 경우 블록이 꼭대기까지 쌓임을 의미하므로 게임 종료 flag를 TRUE로 설정한다.
  - TRUE일 때 **BlockDown()** 함수가 종료되면 프로그램이 종료된다.
- 블록을 필드에 쌓는다
  - **AddBlockToField()**



# 1 초마다 수행되는 동작(BlockDown () 함수)(2/2)

- 완전한 line이 있을 경우 지워주고 점수를 갱신한 뒤 출력한다.
  - `DeleteLine ()` / `PrintScore ()`
- Next 블록을 현재 블록으로 만들어 주고 새로운 next block을 랜덤하게 결정한다.
  - `DrawNextBlock ()`
- 현재 블록의 위치를 초기화 하고 동작을 종료한다.
  - `DrawField ()`



# 함수표

	함 수 이 름	역 할
구현 함수	<code>int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX);</code>	블록이 움직일 수 있는지 확인
	<code>void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX);</code>	예전 블록을 지우고 다시 그려줌
	<code>void BlockDown(int sig);</code>	매초마다 블록을 한 칸씩 내림
	<code>void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX);</code>	블록을 field에 쌓음
	<code>int DeleteLine(char field[HEIGHT][WIDTH]);</code>	채워진 가로줄을 삭제
제공 함수	<code>char menu(void);</code>	화면에 메뉴를 출력함
	<code>void play(void);</code>	테트리스 게임의 play를 수행.
	<code>void DrawField(int currentBlock, int blockRotate, int blockX, int blockY)</code>	테트리스 play를 하기 위한 field를 그려줌
	<code>void DrawNextBlock(int *nextBlock);</code>	next block을 화면에 그림
	<code>void PrintScore(int score);</code>	score를 출력
	<code>void DrawOutline(void);</code>	테트리스 화면의 기본 테두리를 그림
	<code>int GetCommand(void);</code>	key 입력에 대한 command를 return
	<code>int ProcessCommand(int command);</code>	command에 대한 처리
	<code>void DrawBox(int y, int x, int height, int width);</code>	화면에 box를 그려줌
	<code>void DrawBlock(int y, int x, int blockID, int blockRotate, char tile);</code>	(y,x) 좌표에 block에 character tile을 채워서 그려줌

# 시간 복잡도(복습) (1/6)

---

- 다음 insertion sort의 pseudo code를 바탕으로 시간 복잡도를 복습한다.

```
□  INSERTION_SORT(int list[])
□  1  for j ← 2 to length(list)
□  2      do key ← list[j]
□  3      Insert list[j] into the sorted
□  sequence list[1 ... j-1]
□  4      i ← j-1
□  5      while i > 0 and list[i] > key
□  6          do list[i+1] ← list[i]
□  7              i ← i - 1
□  8      list[i+1] ← key
```

## 시간 복잡도(복습) (2/6)

---

### □ 가정

- Input으로 주어진 `list`의 크기는  $n$ 이다. 즉,  $n = \text{length}(\text{list})$ 이다.
- 라인 5의 while loop가 수행되는 횟수는  $j$ 에 의존하므로  $t_j$ 로 나타내기로 하자.
- 라인  $k$ 번째의 statement는 수행되는데 필요한 비용(cost)는  $c_k$ 로 나타낸다.
  - 이 비용이 각 statement마다 다른 이유는 각 statement가 수행하는 동작은 아주 간단한 대입, 덧셈, 뺄셈수행부터 이를 응용하는 곱셈, 나눗셈 등으로 다양하기 때문이다. 같은 비용을 가질 수 없기 때문이다.

## 시간 복잡도(복습) (3/6)

- 다음은 insertion sort의 pseudo code와 비용 및 각 statement를 몇 번 수행하는 지를 보여준다.

Pseudo code	Cost	Times
INSERTION_SORT(int list[])		
1       for j ← 2 to length(list)	$c_1$	$n$
2             do key ← list[j]	$c_2$	$n-1$
3                     Insert list[j] into the sorted sequence list[1 ... j-1]	0	$n-1$
4                     i ← j-1	$c_4$	$n-1$
5             while i > 0 and list[i] > key	$c_5$	$\sum_{j=2}^n t_j$
6                     do list[i+1] ← list[i]	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7                             i ← i - 1	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8             list[i+1] ← key	$c_8$	$n-1$

## 시간 복잡도(복습) (4/6)

- 위의 결과를 종합하면, 총 수행시간  $T(n)$ 은 다음과 같다.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\ + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

- 만약 input list가 이미 정렬되어 있는 상태(best case)
  - 라인 5에서  $i=j-1$ 일 때  $A[i] \leq \text{key}$ 이다. 따라서  $j=2, \dots, n$ 에 대해서  $t_j=1$ 이다.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

- 수행시간은  $an+b$ 의 형태이고, list의 크기는  $n$ 이므로 위 수행시간은  $n$ 의 linear function이다.
  - Constant  $a$ 와  $b$ 는 각 statement의 비용들에 의해서 결정되는 값이다.



# 시간 복잡도(복습) (5/6)

- 만약 input list가 역순으로 정렬되어 있는 상태(worst case)
  - 라인 5에서  $A[j]$ 는 정렬되지 않은 배열  $A[1 \dots j-1]$ 의 모든 원소와 비교되어야 한다. 따라서  $t_j=j$ 이다.

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1, \quad \sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

- 총 수행 시간은 다음과 같다.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left( \frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left( \frac{n(n-1)}{2} \right) + c_7 \left( \frac{n(n-1)}{2} \right) + c_8(n-1) \\ &= \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

- 수행시간은  $an^2+bn+c$ 의 형태이고, list의 크기는  $n$ 이므로 위 수행시간은  $n$ 의 quadratic function이다.
  - Constant  $a$ ,  $b$ 와  $c$ 는 각 statement의 비용들에 의해서 결정되는 값이다.

# 시간 복잡도(복습) (6/6)

---

- Big O notation의 정의
  - $f(x)$ 와  $g(x)$ 를 실수 위에서 정의된 함수라 하자.
  - 만약,  $|f(x)| \leq M|g(x)|$  for all  $x > x_0$  을 만족하는  $M$ 과  $x_0$ 가 존재한다면,
  - $f(x) = O(g(x))$
  
- 이 정의에 의해, 위 부등식을 만족하는  $M$ 과  $x_0$ 가 존재하므로, insertion sort의 시간 복잡도는  $O(n^2)$ 이다.

# 공간 복잡도(복습) (1/1)

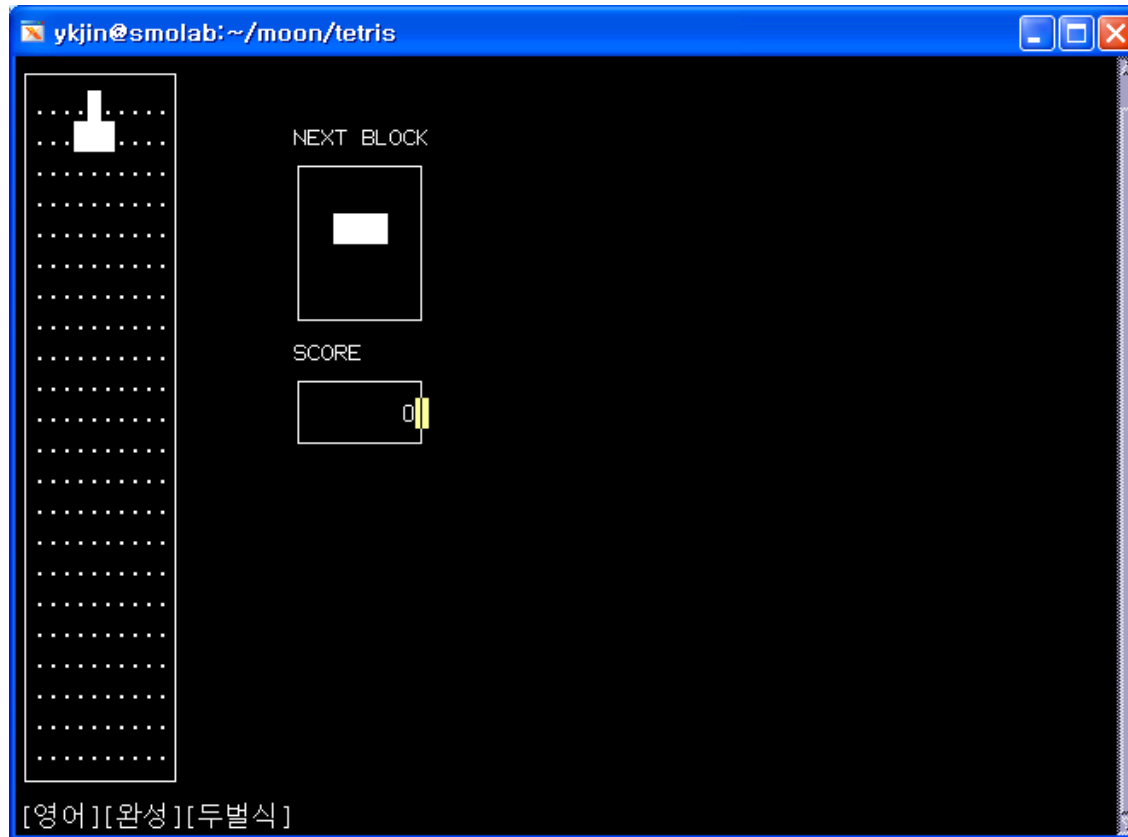
- 다음 insertion sort의 pseudo code를 바탕으로 공간 복잡도를 복습한다.
  
- `INSERTION_SORT(int list[])`
- 1    `for j ← 2 to length(list)`
- 2         `do key ← list[j]`
- 3             `Insert list[j] into the sorted`
- `sequence list[1 ... j-1]`
- 4             `i ← j-1`
- 5         `while i > 0 and list[i] > key`
- 6             `do list[i+1] ← list[i]`
- 7             `i ← i - 1`
- 8         `list[i+1] ← key`
  
- 위에서 `list[]`는  $n$ 개의 크기를 갖고, `i`, `j`, `key`를 저장하기 위한 변수는 고정된 크기를 갖기 때문에, insertion sort의 공간 복잡도는  $O(n)$ 이다

---

# 테트리스 프로젝트 1주차

# 테트리스 게임 프로그램 실행

- 제공된 프로그램 소스를 다음과 같이 compile한 후 실행한다
  - `gcc tetris.c -lncurses`
  - `./a.out`



# global 변수(1/2)

---

## ❑ `char field[HEIGHT][WIDTH]`

- Field의 정보가 저장되는 변수
- 초기 상태에는 쌓여있는 블록이 하나도 없으므로 전부 0으로 초기화

## ❑ `int nextBlock[BLOCK_NUM];`

- 현재 블록의 shape ID와 다음 블록의 ID를 저장하는 변수
  - `nextBlock[0]`: 현재 블록
  - `nextBlock[1]`: 다음 블록
- 고려되는 블록이 2개이므로 `BLOCK_NUM`은 2로 정의
- `rand()` 함수를 사용하여 `[0~6]` 사이의 random한 값으로 초기화

## ❑ `int blockRotate`

- 블록이 몇도 회전 했는가를 나타내는 변수
- 0으로 초기화

## global 변수(2/2)

---

### □ `int blockY`

- 블록의 필드상에서의 y 좌표 값을 저장하는 변수
- 블록을 필드상단에 위치시키기 위하여 y좌표를 -1 로 초기화

### □ `int blockX`

- 블록의 필드상에서의 x 좌표 값을 저장하는 변수
- 블록을 필드중앙에 위치시키기 위하여 x좌표를  $(WIDTH/2)-2$  로 초기화

### □ `int score`

- 플레이어의 점수를 기록
- 0으로 초기화

### □ `int gameOver=0;`

- 게임이 종료되면 1로 set되는 변수

### □ `int timed_out`

- 현재 블록을 매초마다 한 칸씩 떨어뜨리기 위하여 사용
- 0으로 초기화

## 구현할 내용(1/2)

---

- `int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)`
  - 블록이 움직일 좌표와 회전횟수를 이용해 현재 블록이 필드의 해당 위치로 이동할 수 있는지 체크한다.
  
- `void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX)`
  - 블록이 이동할 좌표와 회전횟수, 그리고 command를 이용해 현재 위치의 블록을 지우고, 필드의 새로운 위치에 블록을 그려준다.



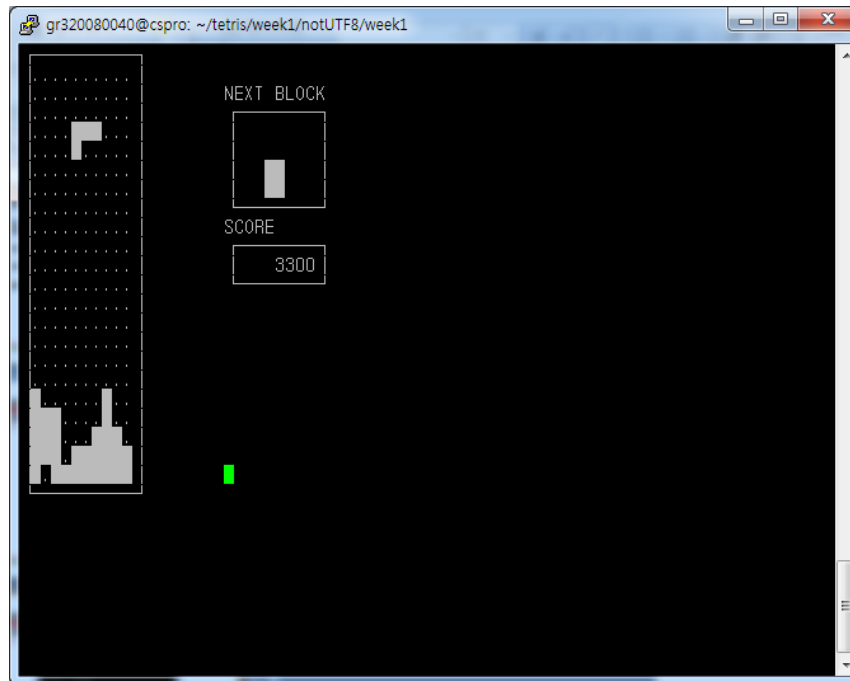
## 구현할 내용(2/2)

---

- `int BlockDown(int sig)`
  - 현재 블록을 매초마다 한 칸씩 아래로 떨어뜨린다.
  - 블록이 더 이상 내려갈 수 없을 경우 블록을 필드에 쌓고 완전히 채워진 line을 지운다.
  
- `void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)`
  - 현재 위치의 현재 블록을 필드에 쌓는다.
  
- `int DeleteLine(char field[HEIGHT][WIDTH])`
  - 완전히 채워진 line을 찾아 지우고 지운 line의 개수에 대한 점수를 return한다.

# 구현결과

- ❑ 블록이 키보드의 입력에 따라 동작한다.
- ❑ 블록이 자동으로 한 칸씩 떨어진다.
- ❑ 블록이 field에 쌓인다.
- ❑ 채워진 Line이 지워지면 score가 갱신된다.
- ❑ Next 블록이 현재 블록으로 바뀐다.



# 테트리스 기본 흐름과의 관계(1/2)

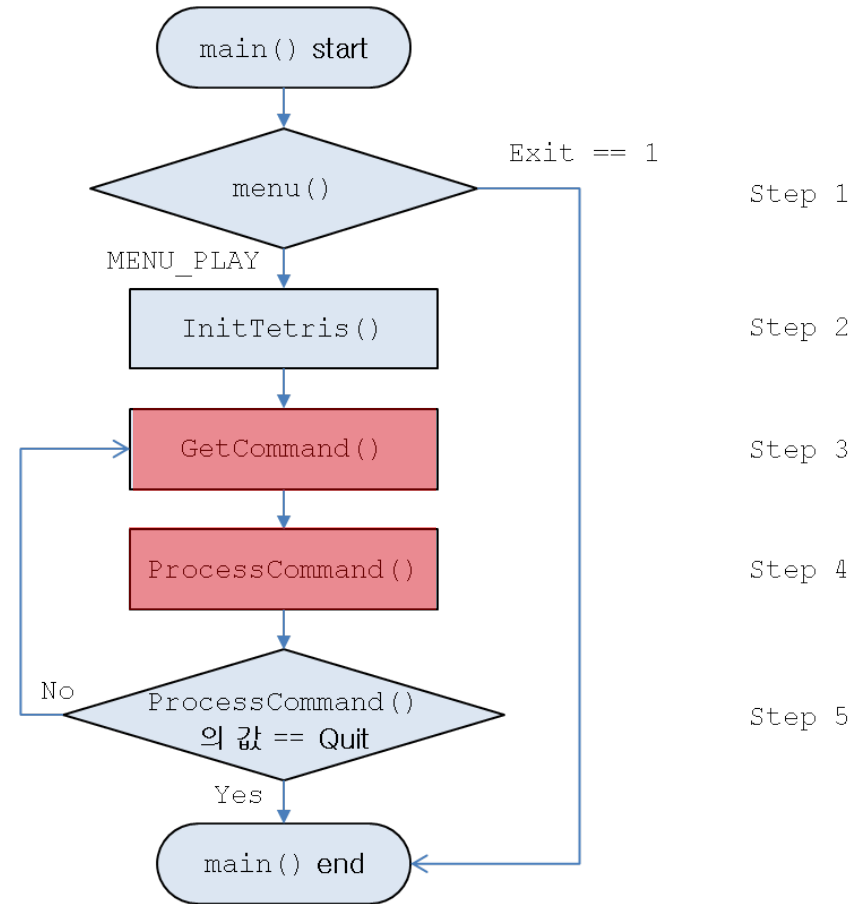
## □ 키입력에 대한 동작

### ● GetCommand()

- 키보드의 입력을 받아 입력에 해당하는 command를 return한다.

### ● ProcessCommand()

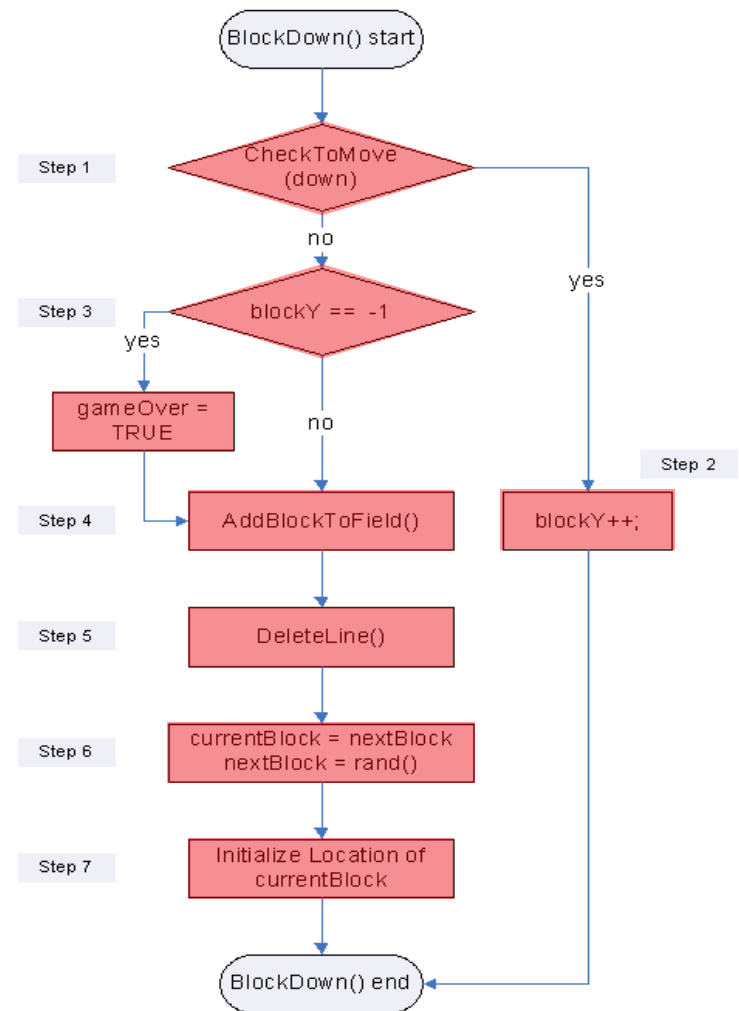
- GetCommand로부터 받은 command를 처리한다.
- **CheckToMove()**
  - Command에 대한 동작이 가능한지 check한다.
- **DrawChange()**
  - Command에 대한 동작이 가능하여 현재 블록의 정보가 수정되면 정보에 따라 현재 블록을 새로 그려준다.



# 테트리스 기본 흐름과의 관계(2/2)

## □ 1초마다 수행되는 동작

- 블록이 한 칸 내려갈 수 있는지 확인한다.
  - **CheckToMove()**
- 내려갈 수 있으면 블록을 아래로 한 칸 내리고 함수를 종료하고, 내려갈 수 없으면 다음 step을 수행한다.
  - **DrawChange()**
- 블록의 y좌표가 초기값인 -1인 경우 블록이 꼭대기까지 쌓임을 의미하므로 게임 종료 flag를 TRUE로 설정한다.
  - TRUE일때 **BlockDown()** 함수가 종료되면 프로그램이 종료된다.
- 블록을 필드에 쌓는다
  - **AddBlockToField()**
- 완전한 line이 있을 경우 지워주고 점수를 갱신한 뒤 출력한다.
  - **DeleteLine()** / **PrintScore()**
- Next 블록을 현재 블록으로 만들어 주고 새로운 next 블록을 랜덤하게 결정한다.
  - **DrawNextBlock()**
- 현재 블록의 위치를 초기화 하고 동작을 종료한다.
  - **DrawField()**



# 함수표

1주차  
구현 함수

	함 수 이 름	역 할
구 현 함 수	<code>int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX);</code>	블록이 움직일 수 있는지 확인
	<code>void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX);</code>	예전 블록을 지우고 다시 그려줌
	<code>void BlockDown(int sig);</code>	매초마다 블록을 한 칸씩 내림
	<code>void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX);</code>	블록을 필드에 쌓음
	<code>int DeleteLine(char field[HEIGHT][WIDTH]);</code>	채워진 가로줄을 삭제
제 공 함 수	<code>char menu(void);</code>	화면에 메뉴를 출력함
	<code>void play(void);</code>	테트리스 게임의 play를 수행.
	<code>void DrawField(int currentBlock, int blockRotate, int blockX, int blockY)</code>	테트리스 play를 하기 위한 필드를 그려줌
	<code>void DrawNextBlock(int *nextBlock);</code>	next 블록을 화면에 그림
	<code>void PrintScore(int score);</code>	score를 출력
	<code>void DrawOutline(void);</code>	테트리스 화면의 기본 테두리를 그림
	<code>int GetCommand(void);</code>	key 입력에 대한 command를 return
	<code>int ProcessCommand(int command);</code>	command에 대한 처리
	<code>void DrawBox(int y, int x, int height, int width);</code>	화면에 box를 그려줌
	<code>void DrawBlock(int y, int x, int blockID, int blockRotate, char tile);</code>	(y,x) 좌표에 블록을 character tile로 채워서 그려줌

# CheckToMove () (1/3)

---

- `int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX )`
  - Parameter : 블록의 움직이고 있는 필드, 이동할 위치와 회전횟수
    - `char field[HEIGHT][WIDTH]`
      - 현재 블록의 움직임을 확인할 테트리스 필드
    - `int currentBlock`
      - 현재 블록의 shape ID
    - `int blockRotate`
      - 블록의 회전 횟수
    - `int blockY`
      - 블록의 y좌표
    - `int blockX`
      - 블록의 x좌표
  - Return
    - 블록이 해당 위치로 이동하거나 회전할 수 있으면 1을, 아니면 0을 return

## CheckToMove () (2/3)

---

- 블록이 회전하거나 블록을 이동하기 전에 블록의 예상 회전 혹은 예상 좌표로 이동 혹은 회전이 가능한지를 check하여준다.
  
- Ex) 블록을 왼쪽으로 이동하라는 command가 입력되었을 경우 왼쪽으로 이동하게 되면 x좌표가 1 감소하므로 다음과 같이 호출한다.
  - `CheckToMove(field, currentBlock, blockRotate, blockY, blockX-1);`
  - `CheckToMove`는 current block의 예상좌표와 회전정보를 받아 가능한지를 check하여 준다.

# CheckToMove () (3/3)

---

- 현재 블록을 움직일 수 없는 경우
  - 예상된 현재 블록이 필드 바깥으로 나갈 경우
  - 예상 현재 블록이 쌓여 있는 블록과 겹치는 경우
  
- 구현 방법
  - 2중 for loop를 이용한다.
  - `i`는 `0~HEIGHT-1`를 나타내고, `j`는 `0~WIDTH-1` 나타낸다.
  - `block[currentBlock][blockRotate][i][j]==1`일 때
    - `i+blockY`와 `j+blockX`의 범위가 field를 벗어나지 않는지를 check한다.
    - `field[i+blockY][j+blockX]==1`인지를 check한다.



# DrawChange () (1/3)

---

□ `void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX )`

● Parameter

- `char field[HEIGHT][WIDTH]`
  - 현재 블록의 움직임을 확인할 테트리스 필드
- `int command`
  - 사용자의 입력에 대한 command
- `int currentBlock`
  - 현재 블록의 shape ID
- `int blockRotate`
  - 블록의 회전 횟수
- `int blockY`
  - 블록의 y좌표
- `int blockX`
  - 블록의 x좌표

## DrawChange () (2/3)

---

- 블록의 움직임이 성공하였을 경우 호출되는 함수로 현재 블록의 좌표와 command를 안다면 블록의 움직이기 전에 대한 블록의 정보를 알 수 있을 것이다.
- Ex)

	블록의 현재 정보	블록의 이전 정보
command	KEY_RIGHT	
blockRotate	0	0
blockY	3	3
blockX	2	1

## DrawChange () (3/3)

---

- 블록의 정보와 command를 이용하여 바뀌기 전 현재 블록의 정보(위치, 회전횟수)를 얻는다.
  - switch문을 사용한다.
- 이전 current block 정보를 이용하여 이전에 그려진 현재 블록을 화면에서 지운다.
  - 2중 for loop을 사용한다.
- 현재 현재 블록의 정보를 이용하여 현재 블록을 화면에 그려준다.
  - DrawBlock () 함수를 호출한다.
- 블록 출력 후 커서가 필드상에 있으므로 move함수를 이용하여 커서를 필드 밖으로 이동해준다.

# AddBlockToField() (1/2)

---

- ❑ `void AddBlockToField (char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX )`
  - Parameter
    - `char field[HEIGHT][WIDTH]`
      - 현재 블록의 움직임을 확인할 테트리스 필드
    - `int currentBlock`
      - 현재 블록의 shape ID
    - `int blockRotate`
      - 블록의 회전 수
    - `int blockX`
      - 블록의 y좌표
    - `int blockY`
      - 블록의 x좌표
- ❑ 블록의 위치 정보와 회전 정보, 모양 정보 등을 받아 블록의 정보를 `field` 배열에 추가한다.

## AddBlockToField() (2/2)

---

- 2중 for loop을 이용하여
  - `Block[currentBlock][blockRotate][i][j]==1`  
일 경우 `field[blockY+i][blockX+j] = 1`로 설정한다.

# DeleteLine () (1/2)

---

□ `int DeleteLine(char field[HEIGHT][WIDTH])`

- Return

- $(\text{지운 line 개수})^2 \times 100$

□ 완전히 채워진 line을 찾아 지우고 지워진 line위의 모든 `field` 배열의 원소 값을 모두 아래로 지워진 line 수만큼 내려주고 지워진 line에 해당하는 점수를 return 한다.

## DeleteLine () (2/2)

---

- 위에서 아래로 내려가면서(각 줄 마다) 완전히 1로 채워진 line을 찾는다.
  - 채워진 line을 찾을 때는 필드에서 한 줄의 element가 모두 1이어야 한다.
    - 임시로 한 줄이 채워졌는지 검사하는 flag를 local variable로 만들어 check한다.
  - 찾으면(flag가 check된 경우) 지워진 line 바로 위에서부터, 필드에 쌓인 블록의 정보를 한 줄씩 내려준다.
    - 프로그램 시 필요한 skill 참고
  
- 점수( (지운 line 개수)<sup>2</sup> × 100 )를 return한다.

# BlockDown () (1/2)

---

□ `void BlockDown(int sig)`

- Parameter

- `int sig`

- BlockDown()함수에서 직접적으로 사용되지 않으므로 설명을 생략한다.

□ 매 1초마다 호출되는 함수로서 사용자의 입력과는 상관 없이 프로그램에 의해서 자동으로 호출된다.



## BlockDown () (2/2)

---

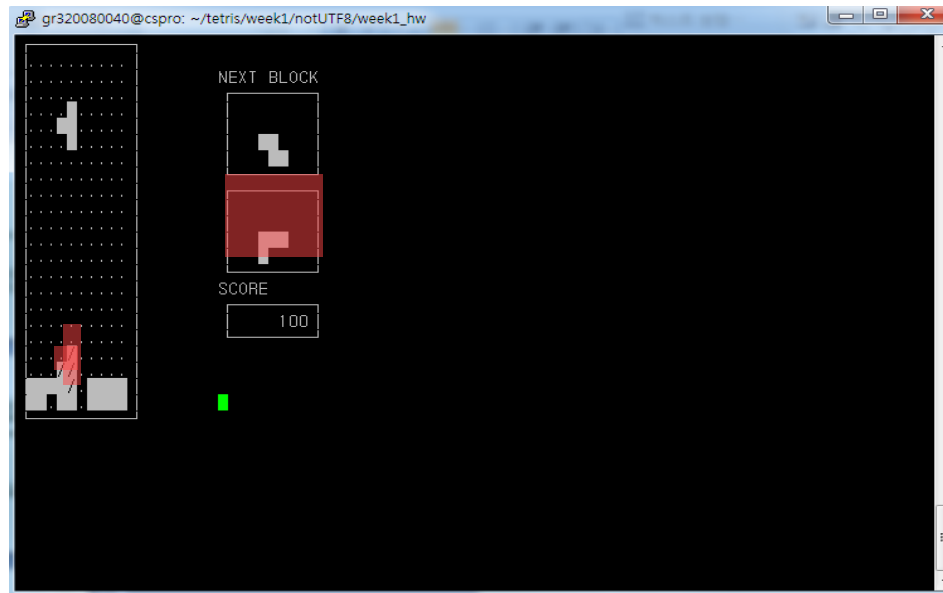
- CheckToMove() 함수를 호출하여 블록이 한 칸 내려갈 수 있는지 확인하고, 내려갈 수 있으면 블록의 y 좌표를 증가시켜주고 block을 한 줄 내린 위치에 다시 그린다. 블록을 더 이상 내릴 수 없을 경우 다음의 동작을 수행한다.
  - 블록의 y좌표가 -1일 경우 gameOver 변수를 1로 setting한다.
  - 블록을 field에 합친다.
    - AddBlockToField() 호출
  - 완전히 채워진 line을 지우고 score를 업데이트 한다.
    - DeleteLine() 호출
  - nextBlock[0]을 nextBlock[1]으로 바꿔준다.
  - nextBlock[1]을 0~6 사이의 random 값으로 설정한다.
  - blockRotate, blockY, blockX를 초기화 한다. ( InitTetris() )에서와 같은 값 )
  - Next 블록을 Next 블록 box안에 그려주고 갱신된 score를 함수를 통하여 화면에 출력한다.
    - DrawNextBlock() / PrintScore()
  - field와 current block을 화면에 갱신하여 출력한다.
    - DrawField()

---

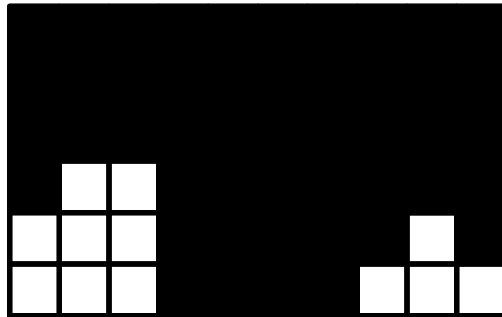
# 테트리스 프로젝트 1주차 숙제

# 구현결과

- 블록이 떨어져서 놓여질 위치가 그림자(/)를 이용해서 표시된다.
- Next block 상자가 하나 더 생기고, 2개의 next block이 예고된다.
- 기존의 지워진 라인 수에 따른 점수 증가에 더하여, 블록이 더 이상 내려갈 수 없을 때(필드 끝까지 내려왔을 때), 블록이 놓여진 위치에서 필드와 블록의 아래 부분이 달은 면적의 수에 비례하여 score를 증가한다.
  - $\text{score} = (\text{지워진 줄 수})^2 * 100$   
+ (블록이 아래에 달은 면적의 수) \* 10

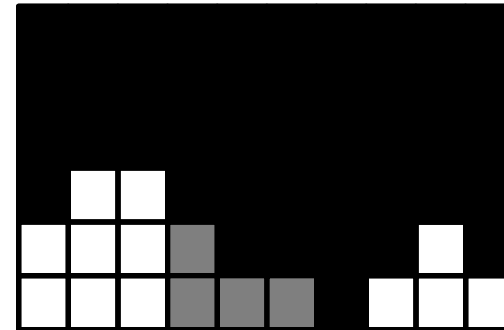


# 구현결과 - score 부분 예제(1/2)



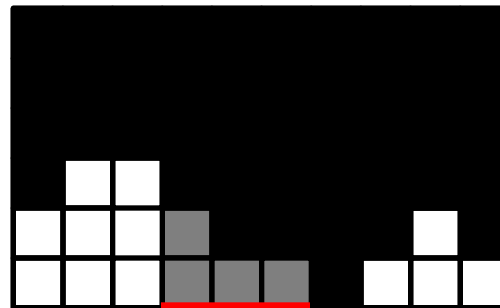
블록이 놓여지기 전

SCORE  
70



블록이 놓여진 후

SCORE  
70



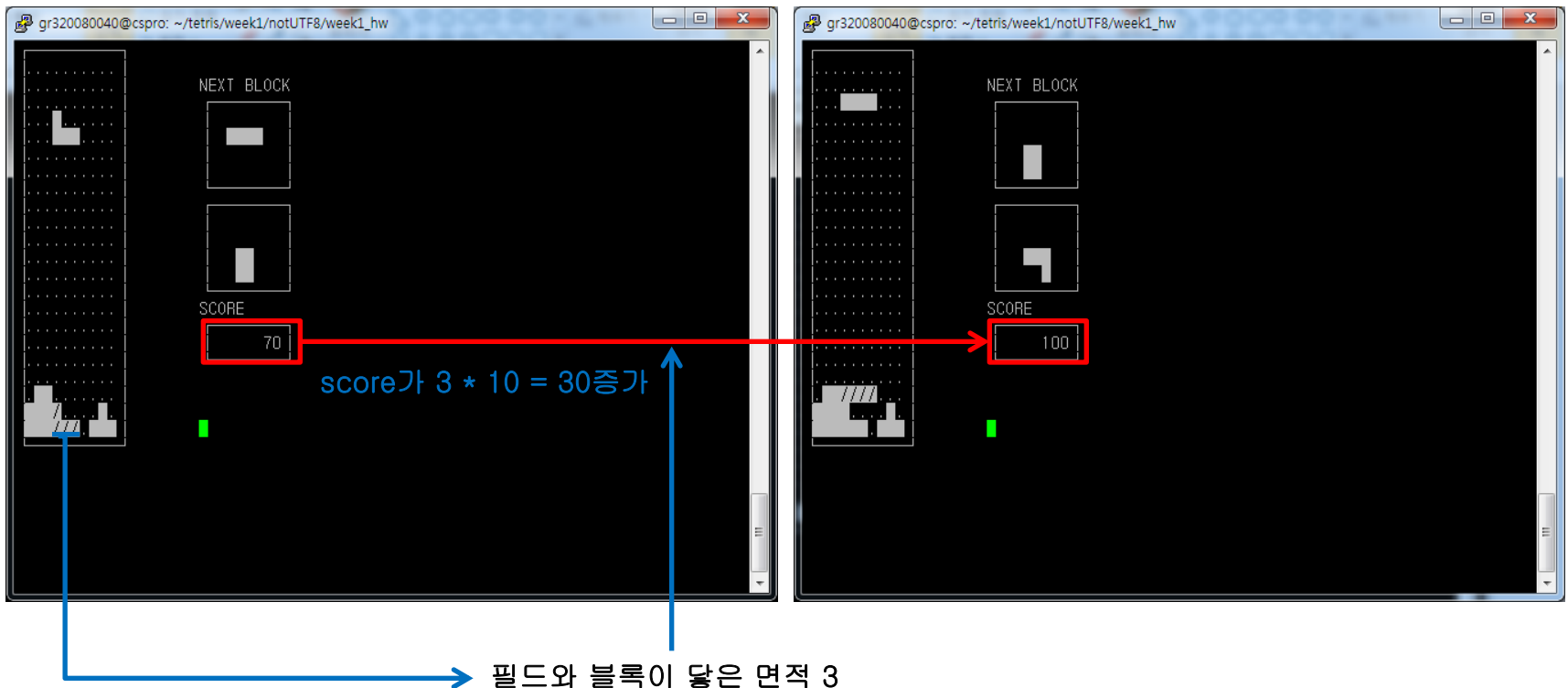
SCORE  
100

필드와 블록이 맞닿은  
면적 : 3칸  
→ 30점 증가

SCORE 30점 증가

## 구현결과 - score 부분 예제(2/2)

- 현재 블록이 필드 위에서 더 이상 내려갈 수 없을 때, 그 블록이 닿은 면적이 3이므로, 30점이 증가한다.



# 그림자 기능(1/2)

---

## □ 구현 및 수정할 함수

- `void DrawBlock(int y, int, x, int blockID, int blockRotate, char tile);`
  - 수정할 함수
- `void DrawShadow(int y, int, x, int blockID, int blockRotate);`
  - 수정할 함수
- `void DrawBlockWithFeatures(int y, int, x, int blockID, int blockRotate);`
  - 추가할 함수

## 그림자 기능(2/2)

---

### □ 함수별 구현 내용

- **DrawBlock()** 함수는 문자(`char tile`)를 parameter로 받아서 블록을 그릴때, `tile`로 블록 안을 채워 그리는 기능을 한다.
  - 현재 블록은 `tile=' '`(space character).
- **DrawShadow()** 함수는 그림자의 위치(현재 블록을 가장 아래로 내렸을 때, 더 이상 내려갈 수 없는 위치)를 찾고, 그 위치에 현재의 블록을 '/'문자를 `tile`로 하여 그리는 함수이다.
- **DrawBlockWithFeatures()** 함수는 두 가지 **DrawBlock()**, **DrawShadow()** 함수를 호출하는 함수로, 기존의 **DrawBlock()** 함수의 위치에 삽입하여 움직임이 갱신될 때마다 현재 블록과 그림자를 함께 그리도록 한다.

## 2개의 블록 미리 보여주기(1/2)

---

### □ 구현 및 수정할 함수

- `void InitTetris();`

- 수정할 함수

- `void DrawNextBlock(int *nextBlock);`

- 수정할 함수

- `void BlockDown(int sig);`

- 수정할 함수

□ 1주차 구현 후 next 블록을 하나 더 추가하기 위해서 `nextBlock` 배열의 `element`의 수를 증가하여 현재 블록을 포함하여 두 개의 next 블록 정보를 저장할 수 있도록 수정해야 한다.



## 2개의 블록 미리보여주기(2/2)

---

### □ 함수 별 구현 내용

- **initTetris()** 함수에서는 배열 형태인 **nextBlock** 변수의 이용해서 두 번째 next 블록 정보(**nextBlock[2]**)를 초기화한다.
- **DrawNextBlock()** 함수는 위에 그린 블록의 초기 위치를 변경하여 2번째 next 블록을 그리는 과정을 추가한다.
- **BlockDown()** 함수에서는 더 이상 아래로 움직일 수 없을 때, 블록 정보를 갱신한다. 즉, 1번째 next 블록이 현재 블록으로, 2번째 next 블록이 1번째 next 블록으로, 마지막으로 2번째 next 블록은 새롭게 생성된다.

# 달은 면적만큼 score 증가하기(1/2)

---

## □ 구현 및 수정할 함수

- `void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX);`
  - 수정할 함수
- `void BlockDown(int sig);`
  - 수정할 함수

## 달은 면적만큼 score 증가하기(2/2)

### □ 함수별 구현 내용

- **AddBlockToField()** 함수에서는 정수형 **touched**라는 변수(달은 면적을 count하는 변수, 다른 변수명도 상관없음)를 만들고, 블록 정보와 필드 정보를 사용해서 현재 블록과 필드가 맞닿아 있는 필드의 면적을 count하고, score를 계산해서 return한다.
  - 더 이상 블록이 내려갈 수 없을 때, 블록을 필드에 추가하는 과정에서 필드에 추가되는 위치의 바로 아래에 필드가 채워져 있는지를 검사한다.
    - 만약 채워져 있다면(1이라면), `touched++`
  - `score = touched * 10;`
- **BlockDown()** 함수에서는 **AddBlockToField()** 함수에서 계산되어 return된 score를 누적한다.

# 테트리스 1주차 실습 결과보고서

---

- 실습 시간에 작성한 프로그램의 함수들이 예비보고서에서 작성한, 각 구현 함수들의 pseudo code와 어떻게 달라졌는지 설명하고, 시간 및 공간 복잡도를 보이시오.
- 테트리스 프로젝트 1주차 숙제 문제를 해결하기 위한 pseudo code를 기술하고, 시간 및 공간 복잡도를 보이시오.

# 테트리스 2주차 실습 예비보고서

---

- 2주차 실습에 구현하는 랭킹 시스템에 대한 자료를 읽어보고, 이를 구현하기 위한 다양한 자료구조를 2가지 이상 생각한다.
- 생각한 각 자료구조에 대해서 새로운 랭킹을 삽입 및 삭제하기 위해 필요한 시간 및 공간 복잡도를 계산한다.
- 생각한 각 자료구조에 대해서 어떻게 정렬된 랭킹( $x \sim y$ 위,  $x \leq y$ ,  $x, y$ 는 정수)을 얻을 수 있을지에 대해서 생각해보고, 그 방법에 대해서 기술하시오.