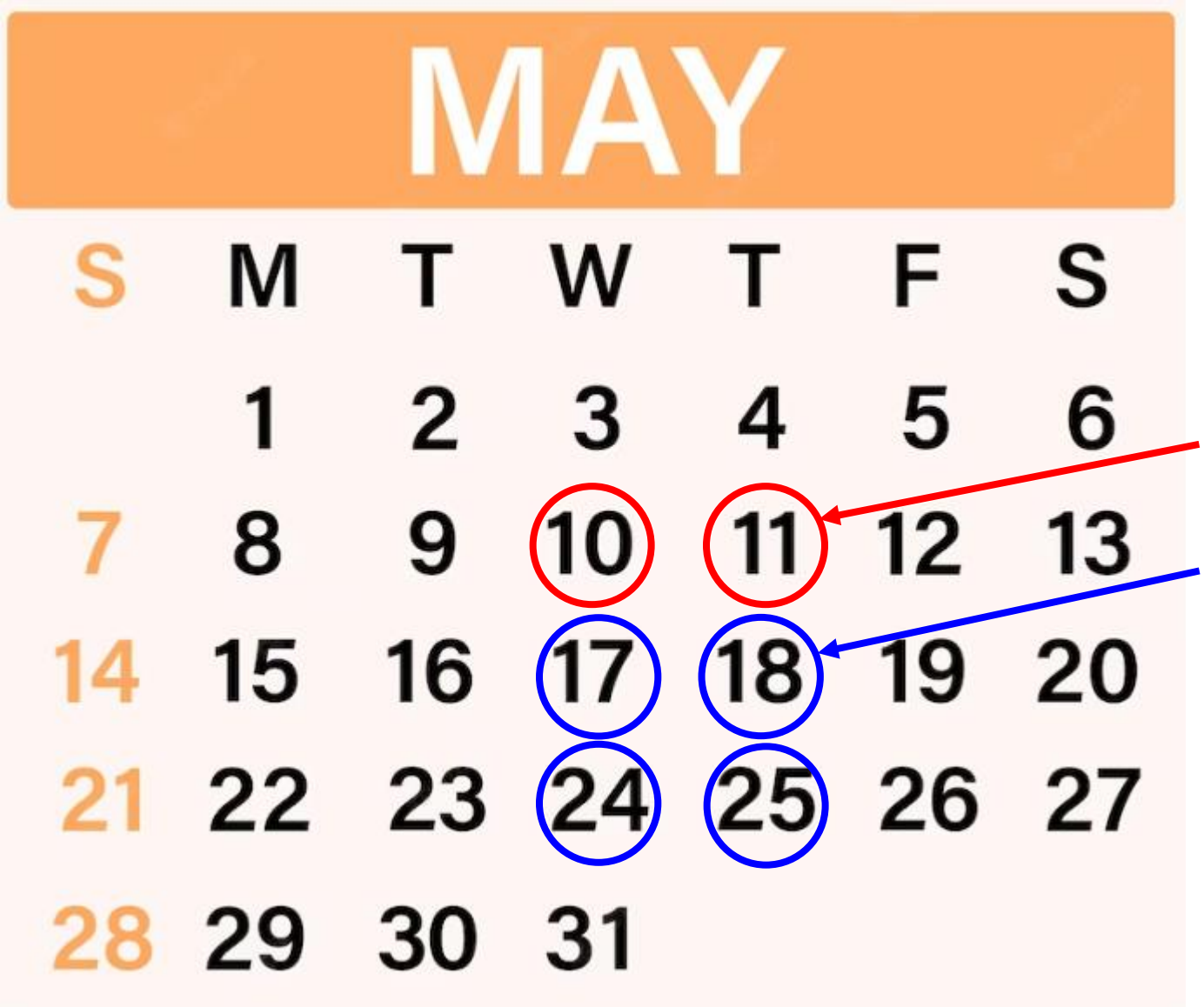
Plug-in Tetris Class 만들기

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

주차별 강의 내용

1. ~~교과목 개요~~
2. ~~C/C++ 기초~~
3. ~~구조체와 클래스~~
4. ~~생성자와 벡터 → 과제#1~~
5. ~~연산자 오버로딩과 프렌즈~~
6. ~~포인터와 동적 배열~~
7. ~~상속 → 과제#2~~
8. ~~중간고사(4/20일 pm7:30)~~
9. ~~다형성과 가상 함수~~, 템플릿 → 과제#3
10. 연결된 자료구조들
11. 예외 처리
12. 표준 템플릿 라이브러리(STL) → 과제#4
13. 분할 컴파일과 네임스페이스
14. 스트림과 파일 입출력
15. 기말고사 (6/8일 pm7:30)

휴보강 일정: (가)반


MAY
S M T W T F S
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
휴강
150분
수업

휴보강 일정: (나)반


MAY
S M T W T F S
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31 1
휴강
150분
수업

accept 함수의 hardcoding?

```
159
160   if (state == TetrisState::Finished)
161     return state;                                    ... }
162
163   else if (state == TetrisState::NewBlock) {
194   else if (state == TetrisState::Running) {
195
196     state = TetrisState::Running;
197     bool touchDown = false;
198     Matrix *tempBlk, *tempBlk2;
199
200     switch (key) { // perform the requested action
201       case 'a': left--; break;
202       case 'd': left++; break;
203       case 'w':
204         degree = (degree + 1) % numDegrees;
205         currBlk = setOfBlockObjects[type][degree];
206         break;
207       case 's': top++; break;
208       case ' ':
209         while (true) {
210           top++;
211           tempBlk = iScreen->clip(top, left, top + currBlk->get_dy(), left + currBlk->get_dx());
212           tempBlk2 = tempBlk->add(currBlk);
213           delete tempBlk;
214           if (tempBlk2->anyGreaterThan(1)) {
215             delete tempBlk2;
216             break;
217           }
218           delete tempBlk2;
219         }
220         break;
221       default: cout << "Tetris::accept: wrong key input" << endl;
222     }
223
```

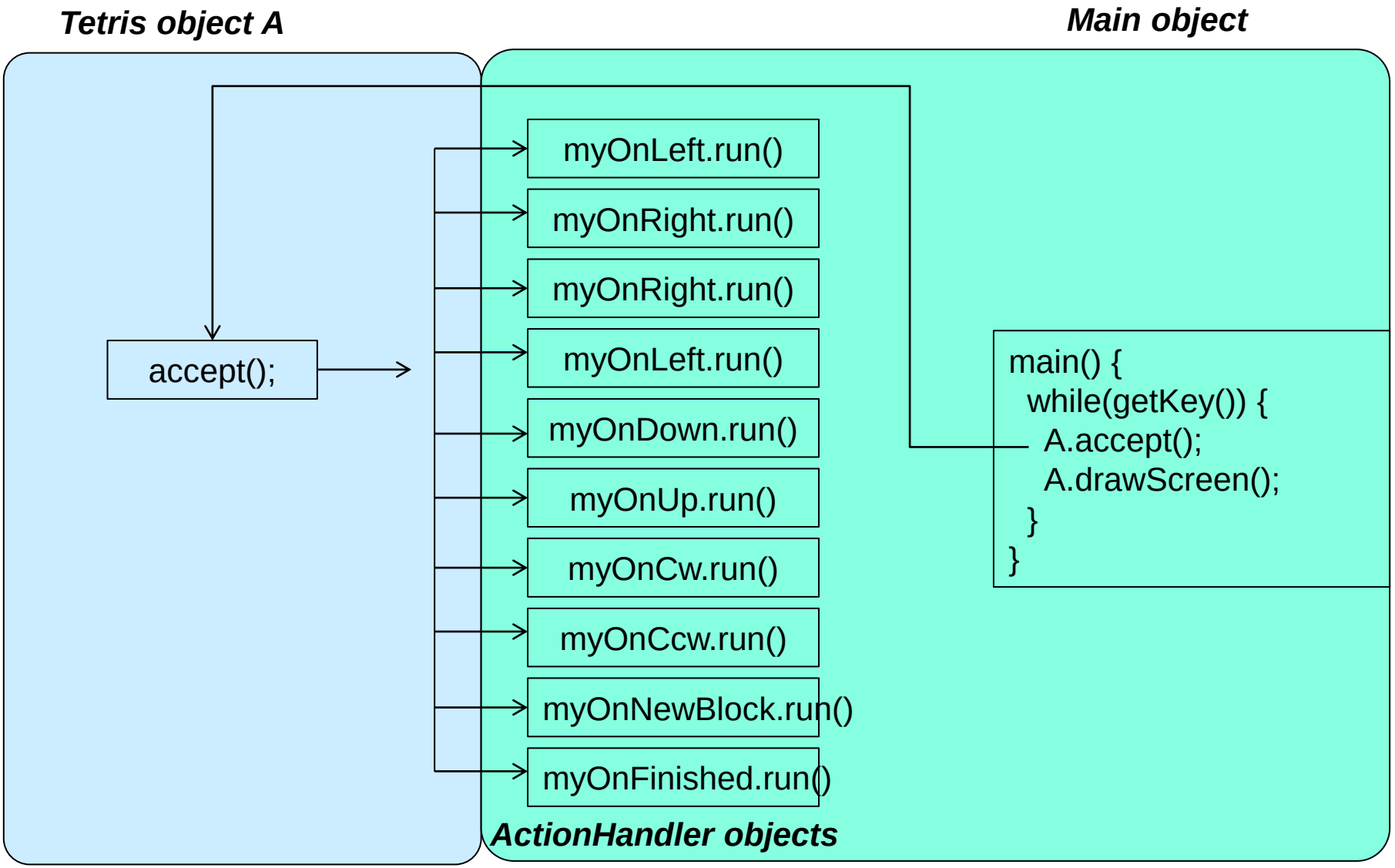
accept 함수의 hardcoding?

```
159
160   if (state == TetrisState::Finished)
161     return state;                                   ... }
162
163   else if (state == TetrisState::NewBlock) {
194   else if (state == TetrisState::Running) {
195
196     state = TetrisState::Running;
197     bool touchDown = false;
198     Matrix *tempBlk, *tempBlk2;
199
200     switch (key) { // perform the requested action
201       case 'a': left--; break;
202       case 'd': left++; break;
203       case 'w':
204         degree = (degree + 1) % numDegrees;
205         currBlk = setOfBlockObjects[type][degree];
206         break;
207       case 's': top++; break;
208       case ' ':
209         while (true) {
210           top++;
211           tempBlk = iScreen->clip(top, left, top + currBlk->get_dy(), left + currBlk->get_dx());
212           tempBlk2 = tempBlk->add(currBlk);
213           delete tempBlk;
214           if (tempBlk2->anyGreaterThan(1)) {
215             delete tempBlk2;
216             break;
217           }
218           delete tempBlk2;
219         }
220         break;
221       default: cout << "Tetris::accept: wrong key input" << endl;
222     }
223
```

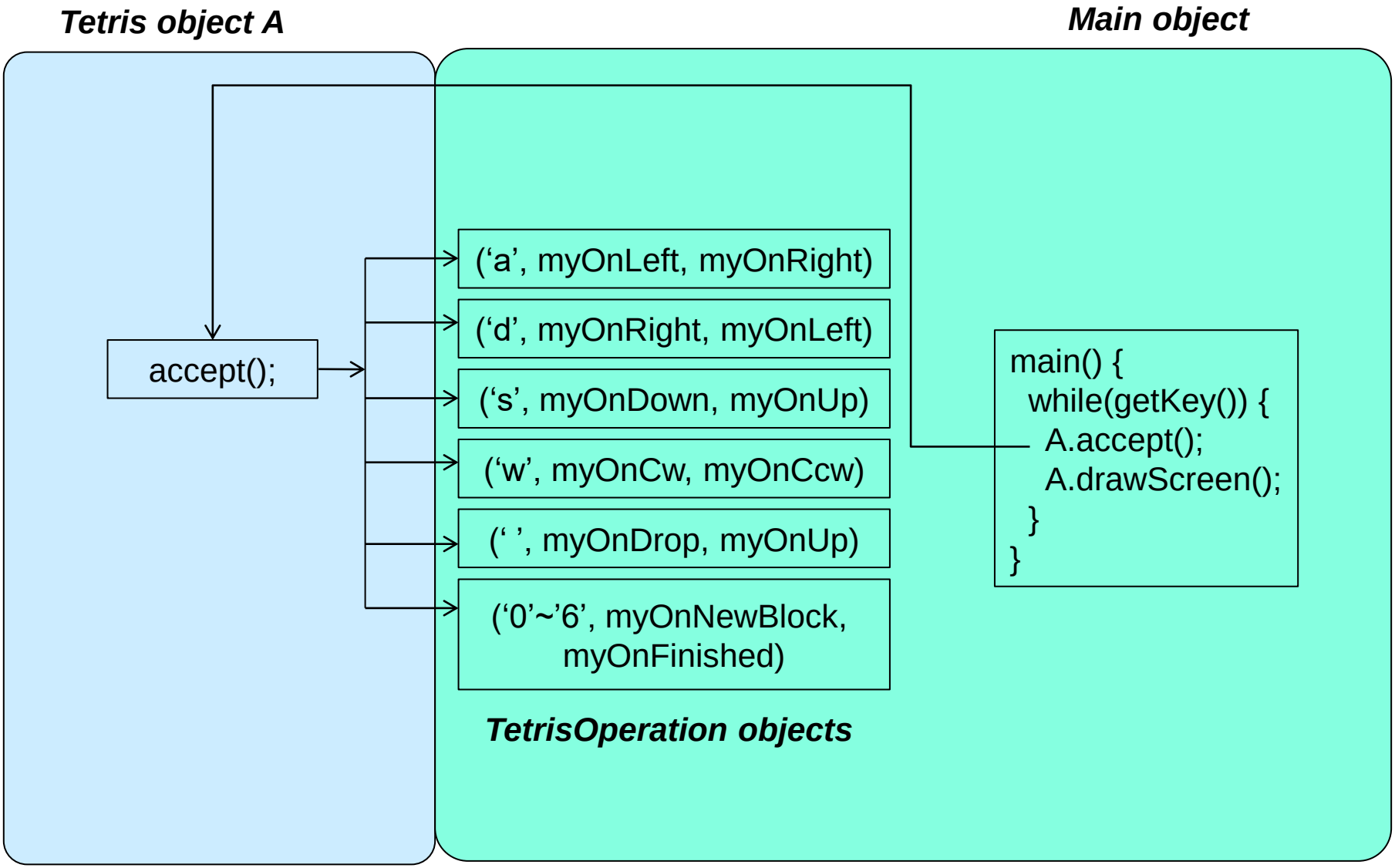
hardcoded constant:
- why 'a'?

hardcoded logic:
- why move by 1 unit?
- why 5 operations?
- why only 's' and SPC produce a new block?

Plug-in Tetris (version 1)



Plug-in Tetris (version 2)



Plug-in Tetris

- 예시 코드

```
class MyOnLeft : public ActionHandler {
public: void run(Tetris *t, char key) { t->left-- }
};
class MyOnRight : public ActionHandler {
public: void run(Tetris *t, char key) { t->left++ }
};
Tetris::init(setOfBlockArrays, 7, 4);
Tetris::setOperation('a', Running, new MyOnLeft(), Running, new
MyOnRight(), Running);
Tetris *board = new Tetris(10, 10);
char key = (char) ('0' + rand()%7);
board->accept(key);
drawScreen(board->get_oScreen(), board->get_wallDepth());
```

General Tetris Class

- 확장 가능한 Tetris 클래스 설계 (부제: 조립식 코딩)
  - 클래스 내부의 Data를 외부에서 변경함 → Tetris::init()
  - 클래스 내부의 Logic을 외부에서 변경함 → 이번 강의 내용
  - 클래스 정의의 구체화 → 클래스 확장 범위를 한계 짓기 위함

    : "**Tetris 게임은 한정된 2차원 영역 안에서 임의 모양의 블록을 대상으로 주어진 key 마다 정해진 action 을 수행하고, 충돌 발생시 counteraction 을 수행하며 오래 버티는 게임**"

- 문법
  - virtual function 이 언제 필요한가?

Main.cpp

- main 함수에서 plug-in 할 테트리스 동작들

```
180 class MyOnLeft : public ActionHandler {
181 public:
182     void run(Tetris *t, char key) {
183         t->left = t->left - 1;
184         return;
185     }
186 };
187
188 class MyOnRight : public ActionHandler {
189 public:
190     void run(Tetris *t, char key) {
191         t->left = t->left + 1;
192         return;
193     }
194 };
195
```

Main.cpp

- Main.cpp.v5 파일에 빨간 박스만 추가된 파일임
  - 빨간 박스는 plug-in 할 테트리스 동작들과 그것을 기반으로 하는 상태 기계를 자유롭게 정의하는 부분임

```
196 int main(int argc, char *argv[]) {
197   char key;
198   registerAlarm(); // register one-second timer
199   srand((unsigned int)time(NULL)); // init the random number generator
200
201   TetrisState state;
202   Tetris::init(setOfBlockArrays, MAX_BLK_TYPES, MAX_BLK_DEGREES);
203
204   ////////////////////////////////////////////////////////////////////////
205   /// Plug-in architecture for generalized Tetris class
206   ////////////////////////////////////////////////////////////////////////
207   Tetris::setOperation('a', TetrisState::Running, new MyOnLeft(),  TetrisState::Running, new MyOnRight(), TetrisState::Running);
208   Tetris::setOperation('d', TetrisState::Running, new MyOnRight(), TetrisState::Running, new MyOnLeft(),  TetrisState::Running);
209   Tetris::setOperation('s', TetrisState::Running, new OnDown(),    TetrisState::Running, new OnUp(),       TetrisState::NewBlock);
210   Tetris::setOperation('w', TetrisState::Running, new OnClockWise(), TetrisState::Running, new OnCounterClockWise(),  TetrisState::Ru
211   Tetris::setOperation(' ', TetrisState::Running, new OnDrop(),    TetrisState::Running, new OnUp(),       TetrisState::NewBlock);
212   Tetris::setOperation('0', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
213   Tetris::setOperation('1', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
214   Tetris::setOperation('2', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
215   Tetris::setOperation('3', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
216   Tetris::setOperation('4', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
217   Tetris::setOperation('5', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
218   Tetris::setOperation('6', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
219   ////////////////////////////////////////////////////////////////////////
220
221   Tetris *board = new Tetris(10, 10);
222   key = (char) ('0' + rand() % board->get_numTypes());
223   board->accept(key);
224   drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
225
```

Main.cpp

```
221   Tetris *board = new Tetris(10, 10);
222   key = (char) ('0' + rand() % board->get_numTypes());
223   board->accept(key);
224   drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
225
226   while ((key = getch()) != 'q') {
227     state = board->accept(key);
228     drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
229     if (state == TetrisState::NewBlock) {
230       key = (char) ('0' + rand() % board->get_numTypes());
231       state = board->accept(key);
232       drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
233       if (state == TetrisState::Finished)
234         break;
235     }
236   }
237
238   delete board;
239
240   Tetris::deinit();
241   cout << "(nAlloc, nFree) = (" << Matrix::get_nAlloc() << ',' << Matrix::get_nFree() << ")" << endl;
242   cout << "Program terminated!" << endl;
243   return 0;
244 }
```

Tetris.cpp

```
1  #include "Tetris.h"
2
3  using namespace std;
4
5  ///************************************************************
6  /// static member variables and functions
7  ///************************************************************
8  // TetrisOperation 때문에 Tetris 클래스에 추가된 코드들
9  /// Tetris Operation related
10 int Tetris::nOps = 0;
11 TetrisOperation *Tetris::operations[MAX_TETRIS_OPERATIONS];
12
13 int Tetris::findOpIdxByKey(char key) {
14   for (int id = 0; operations[id] != NULL; id++) {
15     if (operations[id]->key == key)
16       return id;
17     }
18   return -1;
19 }
20
21 void Tetris::setOperation(char key, TetrisState s0, ActionHandler *h1,
22                           TetrisState s1, ActionHandler *h2, TetrisState s2) {
23   int idx = findOpIdxByKey(key);
24   if (idx >= 0) {
25     delete operations[idx];
26     operations[idx] = new TetrisOperation(key, s0, h1, s1, h2, s2);
27   }
28   else {
29     if (nOps == MAX_TETRIS_OPERATIONS) {
30       cerr << "Tetris::operations[] is full." << endl;
31       return;
32     }
33     operations[nOps] = new TetrisOperation(key, s0, h1, s1, h2, s2);
34     nOps++;
35   }
36 }
37
```

Tetris.cpp

```
38  void Tetris::setDefaultOperations(void) {
39    setOperation('a', TetrisState::Running, new OnLeft(),  TetrisState::Running, new OnRight(), TetrisState::Running);
40    setOperation('d', TetrisState::Running, new OnRight(), TetrisState::Running, new OnLeft(), TetrisState::Running);
41    setOperation('s', TetrisState::Running, new OnDown(),  TetrisState::Running, new OnUp(), TetrisState::NewBlock);
42    setOperation('w', TetrisState::Running, new OnClockWise(), TetrisState::Running, new OnCounterClockWise(), TetrisState::Running);
43    setOperation(' ', TetrisState::Running, new OnDrop(),  TetrisState::Running, new OnUp(), TetrisState::NewBlock);
44    setOperation('0', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
45    setOperation('1', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
46    setOperation('2', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
47    setOperation('3', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
48    setOperation('4', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
49    setOperation('5', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
50    setOperation('6', TetrisState::NewBlock, new OnNewBlock(), TetrisState::Running, new OnFinished(), TetrisState::Finished);
51  }
52
53  /// Tetris game related
54  Matrix *** Tetris::setOfBlockObjects = NULL;
55  int Tetris::numTypes = 0;
56  int Tetris::numDegrees = 0;
57  int Tetris::wallDepth = 0;
58
59  void Tetris::init(int **setOfBlockArrays, int nTypes, int nDegrees) {
60    if (nOps == 0)
61      setDefaultOperations();
62
63    if (setOfBlockObjects != NULL) // already allocated?
64      deinit();
65
```

// TetrisOperation 때문에 Tetris 클래스에 추가된 코드

Tetris.cpp

```
213  /// accessors
214  Matrix *Tetris::overlap_currBlk(void) {
215    Matrix *tBlk1, *tBlk2;
216    tBlk1 = iScreen->clip(top, left, top + currBlk->get_dy(), left + currBlk->get_dx());
217    tBlk2 = tBlk1->add(currBlk);
218    delete tBlk1;
219    return tBlk2;
220  }
221
222  /// mutators
223  void Tetris::update_oScreen(Matrix *tempBlk, int y, int x) {
224    oScreen->paste(iScreen, 0, 0);
225    oScreen->paste(tempBlk, y, x);
226  }
227
228  TetrisState Tetris::accept(char key) {
229    int idx = findOpIdxByKey(key);
230    if (idx == -1) {
231      cout << "unknown key! (int=" << (int) key << ")" << endl;
232      return state;
233    }
234    TetrisOperation *op = operations[idx];
235    if (state != op->preState) {
236      cout << "wrong preState for the current key!" << endl;
237      return state;
238    }
239    op->hAction->run(this, key);
240    Matrix *tempBlk = overlap_currBlk();
241    if (anyConflict(tempBlk) == false) {
242      state = op->postAState;
243    }
244    else {
245      op->hCounterAction->run(this, key);
246      delete tempBlk;
247      tempBlk = overlap_currBlk();
248      state = op->postCState;
249    }
250    update_oScreen(tempBlk, top, left);
251    delete tempBlk;
252    return state;
253  }
```

```
142  bool anyConflict(Matrix *tempBlk) {
143    return tempBlk->anyGreaterThan(1);
144  }
```

// TetrisOperation 들을 기반으로 동작하는 accept 함수

Tetris.h

```
16  extern Matrix *deleteFullLines(Matrix *screen, Matrix *blk, int top, int dw);
17  extern bool anyConflict(Matrix *tempBlk);
18  //extern int *allocArrayScreen(int dy, int dx, int dw);
19  //extern void deallocArrayScreen(int *array);
20
21  class Tetris;
22
23  class ActionHandler {
24  public:     // 가상 함수로 정의된 추상 클래스
25      virtual void run(Tetris *t, char key) = 0;
26  };
27
28  class TetrisOperation {
29  public:
30      char key;
31      ActionHandler *hAction;
32      ActionHandler *hCounterAction;
33      TetrisState preState;
34      TetrisState postAState;
35      TetrisState postCState;
36      TetrisOperation(char ch, TetrisState s0, ActionHandler *h1,
37          TetrisState s1, ActionHandler *h2, TetrisState s2) {
38          key = ch;
39          hAction = h1;
40          hCounterAction = h2;
41          preState = s0;
42          postAState = s1;
43          postCState = s2;
44      }
45      ~TetrisOperation() {
46          delete hAction;
47          delete hCounterAction;
48      }
49  };
50
```

```
1   #pragma once
2   #include <iostream>
3   #include <cstdlib>
4   #include "Matrix.h"
5
6   using namespace std;
7
8   #define MAX_TETRIS_OPERATIONS 100
9
10  enum class TetrisState {
11      NewBlock,
12      Running,
13      Finished,
14  };
15
```

// Action 과 CounterAction 으로 정의된 Tetris Operation 클래스

Tetris.h

```
51  class Tetris {
52  protected:
53      // TetrisOperation related variables
54      static int nOps;
55      static TetrisOperation *operations[MAX_TETRIS_OPERATIONS];
56      static int findOpIdxByKey(char key);
57      static void setDefaultOperations(void);
58
59  public:
60      static void setOperation(char key, TetrisState s0, ActionHandler *h1,
61                               TetrisState s1, ActionHandler *h2, TetrisState s2);
62      // Tetris game related variables
63      static Matrix ***setOfBlockObjects;
64      static int numTypes;
65      static int numDegrees;
66      static int wallDepth;
67      int rows; // rows of screen =   dy + 2*wallDepth
68      int cols; // columns of screen = dx + 2*wallDepth
69      int type;
70      int degree;
71      int top;
72      int left;
73      TetrisState state;
74      Matrix *iScreen;
75      Matrix *oScreen;
76      Matrix *currBlk;
77
78      static void init(int **setOfBlockArrays, int nTypes, int nDegrees);
79      static void deinit(void);
80      Tetris(int cy, int cx);
81      ~Tetris();
82      // accessors
83      static int get_wallDepth(void)  { return wallDepth; }
84      static int get_numTypes(void)   { return numTypes; }
85      Matrix *get_oScreen(void) const { return oScreen; }
86      Matrix *overlap_currBlk(void);
87      // mutators
88      void update_oScreen(Matrix *tempBlk, int y, int x);
89      TetrisState accept(char key);
90  };
```

// TetrisOperation 때문에 Tetris 클래스에 추가된 코드들

// accept 함수를 TetrisOperation 단위로 구성하기 위해서 필요한 단위 함수들

Tetris.h

- ActionHandler 예시들

```
92  ////////////////////////////////////////////////////////////////////
93  /// Examples of ActionHandlers
94  ////////////////////////////////////////////////////////////////////
95
96  class OnLeft : public ActionHandler {
97  public:
98      void run(Tetris *t, char key) {
99          t->left = t->left - 1;
100         return;
101     }
102 };
103
104 class OnRight : public ActionHandler {
105 public:
106     void run(Tetris *t, char key) {
107         t->left = t->left + 1;
108         return;
109     }
110 };
111
112 class OnDown : public ActionHandler {
113 public:
114     void run(Tetris *t, char key) {
115         t->top = t->top + 1;
116         return;
117     }
118 };
119
120 class OnUp : public ActionHandler {
121 public:
122     void run(Tetris *t, char key) {
123         t->top = t->top - 1;
124         return;
125     }
126 };
127
```

Tetris.h

- ActionHandler 예시들

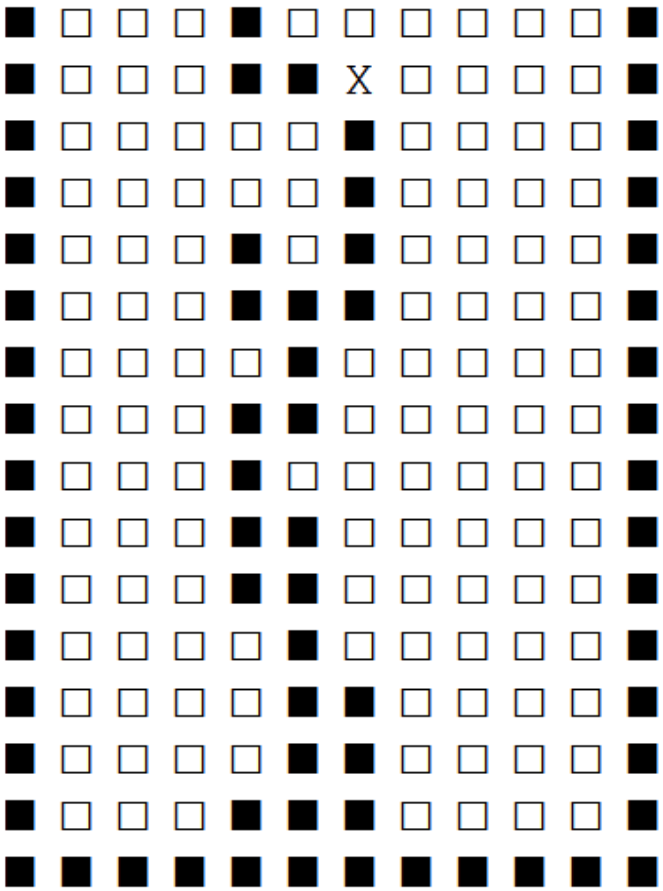
```
128 class OnDrop : public ActionHandler {
129 public:
130     void run(Tetris *t, char key) {
131         Matrix *tempBlk;
132         while (true) {
133             t->top = t->top + 1;
134             tempBlk = t->overlap_currBlk();
135             if (anyConflict(tempBlk) == true) {
136                 delete tempBlk;
137                 break;
138             }
139             delete tempBlk;
140         }
141         return;
142     }
143 };
144
145 class OnClockWise : public ActionHandler {
146 public:
147     void run(Tetris *t, char key) {
148         t->degree = (t->degree + 1) % t->numDegrees;
149         t->currBlk = t->setOfBlockObjects[t->type][t->degree];
150         return;
151     }
152 };
153
154 class OnCounterClockWise : public ActionHandler {
155 public:
156     void run(Tetris *t, char key) {
157         t->degree = (t->degree + 3) % t->numDegrees;
158         t->currBlk = t->setOfBlockObjects[t->type][t->degree];
159         return;
160     }
161 };
162
```

Tetris.h

- ActionHandler 예시들

```
163 class OnNewBlock : public ActionHandler {
164 public:
165     void run(Tetris *t, char key) {
166         if (t->currBlk != NULL) // why test currBlk != NULL?
167             t->oScreen = deleteFullLines(t->oScreen, t->currBlk, t->top, t->wallDepth);
168         t->iScreen->paste(t->oScreen, 0, 0);
169         // select a new block
170         t->type = key - '0';
171         t->degree = 0;
172         t->top =  t->wallDepth;
173         t->left = t->cols/2 - t->wallDepth/2;
174         t->currBlk = t->setOfBlockObjects[t->type][t->degree];
175         return;
176     }
177 };
178
179 class OnFinished : public ActionHandler {
180 public:
181     void run(Tetris *t, char key) {
182         cout << "OnFinished.run() called" << endl;
183         return;
184     }
185 };
```

실행 결과 (이전과 동일함)



좋은 코드의 특징

- 가독성
  - 하나의 변수는 하나의 의미만, 하나의 함수는 하나의 기능만을 표현
  - 변수와 함수 이름이 서술적일수록 좋음 (예: "변수 = 동사(변수)")
- 모듈화
  - 가정(assumption)과 파생 로직의 분리(예: hardcoded constants)
  - 인터페이스와 내부 구현의 명확한 분리
- 간결성
  - 중복된 로직이 없고 필수적인 최소 로직만 존재
- 확장성
  - Generic code 와 plugin code 의 분리
- 이식성
  - 시스템 함수들과 사용자 함수들의 명확한 분리
- 성능
  - 성능 최적화가 필요한 common case가 코드 상에서 쉽게 식별될 수 있으면 충분함

코딩 숙제: TenTrix

- Tetris 게임의 한 가지 변형
- Main.cpp 파일 안에서 Tetris 클래스에 TetrisOperation 을 결합하는 방식으로 구현하기 바람: Tetris.h (v10) 와 Tetris.cpp (v10) 을 수정해서는 안 됨
- 키 동작은 다음과 같이 정의됨
  - 'a' : 왼쪽으로 한 칸 이동함
  - 'd' : 오른쪽으로 한 칸 이동함
  - 's' : 아래쪽으로 한 칸 이동함
  - 'e' : 위쪽으로 한 칸 이동함
  - 'w' : 시계방향으로 90도 회전함

  → 위 키들은 충돌을 일으켜도 XX로 표시하고, 그 동작을 되돌리지 않음. 단, currBlk 이 screen 영역을 벗어나서는 안 됨.
  - ' ' : currBlk 을 현재 위치에서 screen 에 고정함. 단 충돌이 있을 경우에는 고정되지 않음. 하나의 수평 라인 또는 수직 라인이 가득차면, 그 라인은 공백으로 대체됨. 이 때 공백 라인을 다른 라인으로 채우지 않음!! 이를 위해 myDeleteFullLines() 함수를 작성할 것.
  - '0' ~ '6' : 해당 index 의 블록이 화면 상단에 출현함. 출현 즉시 충돌하면 게임 종료함