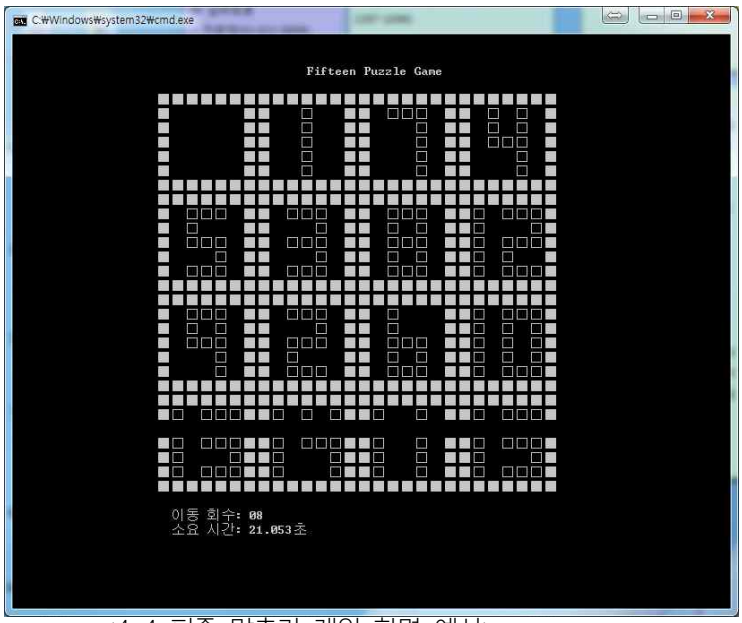
보 고 서 #3

제 목: 4x4 퍼즐 게임 구현

과 목 명:	고급프로그래밍및실습(가)
학 과:	스마트시스템소프트웨어학과
학 번:	20170404
이 름:	한종수
제 출 일:	2018년 5월 16일(수)
담당교수:	한 영 준

보고서 3(제목: 4x4 퍼즐 게임 구현)

1. 보고서 3는 수업시간에 다룬 C++ “동적할당” 과 “연산자 오버로딩” 기능을 보고서 2를 통해 학습해보는 과정이다. 첨부된 puzzleGame.cpp에 주어진 클래스와 미완성 C++ 코드를 완성하여 아래 그림과 같이 4x4 퍼즐 맞추기 콘솔 응용 프로그램을 완성한다. 다음 조건을 만족하도록 게임을 구현한다.



<4x4 퍼즐 맞추기 게임 화면 예시>

(1) 처음에 퍼즐 조각들이 제 위치에 있어야 하고 게임이 시작되면 주어진 반복 회수만큼 섞는 과정을 수행한다. 이때에 퍼즐이 섞이는 과정을 화면에 확인할 수 있도록 시간지연을 준다. 시간 지연을 위해 <windows.h>에 선언된 Sleep() 함수를 사용한다.
(2) 퍼즐이 섞이면 퍼즐 조각의 이동을 위해 화살표 키들을 사용하여 퍼즐을 이동시킨다.
(3) 모든 퍼즐들의 조각들이 제자리를 찾으면 게임이 종료된다. 물론 ESC키를 눌러도 게임을 종료되지만 반복회수나 시간 값을 0으로 화면에 출력한다.
(4) 퍼즐을 맞추기 위해 퍼즐 조각을 움직인 회수와 걸리는 시간을 계산하여 퍼즐 이동시마다 화면에 출력한다.

2. 주의

(1) 첨부된 puzzleGame.cpp 파일의 C++ 소스를 충분히 검토 후 puzzleGame 클래스를 수정하거나 추가적인 변수나 함수를 포함할 수 있으나 최대한 기존 코드를 변경하지 않고 최대한 활용하여 C++ 코드를 완성한다.
(2) 인터넷에 유사한 소스를 참조하더라도 본인이 이해하고 스스로 게임을 구현하는 하는 경우에 본인 창작물로 인정받을 수 있으나 다른 학생의 소스나 인터넷에 있는 소스와 유사도가 높으면 감정 당할 수 있으니 주의하기 바랍니다. 소스코드의 길지 않기 때문에 스스로 구현해서 제출하기 바랍니다.

3. 제출일

(1) 가반: 2018년 5월 16일 수요일 수업시간 전까지 제출
(2) 나반: 2018년 4월 18일 금요일 수업시간 전까지 제출

4. 제출 항목

(1) 주석을 포함한 C++ 소스 코드를 프린트로 출력하여 제출
(2) 게임 시작 단계, 게임 도중 단계, 게임 종료 단계의 화면을 이미지 캡쳐하여 프린트 출력하여 제출
(3) 첨부된 보고서 표지를 반드시 사용함

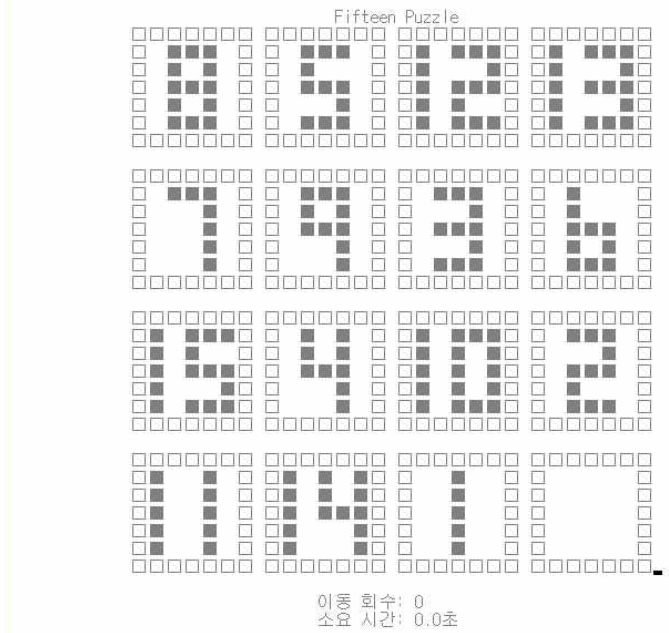
```cpp
#include <iostream>
#include <fstream>
#include <windows.h> //GetStdHandle(), Sleep(), SetConsoleCursorPosition() 함수 사용
#include <conio.h> //getch()와 kbhit() 함수 사용
#include <ctime> //clock() 함수 사용

using namespace std;

#define ESC 27 //게임 종료
#define LEFT 75 //왼쪽 화살표 키 ASCII값
#define RIGHT 77 // 오른쪽 화살표 키 ASCII값
#define DOWN 80 //아래 화살표 키 ASCII값
#define UP 72 //위쪽 화살표 키 ASCII값

#define MDIM 4 //4x4 퍼즐맵의 크기
#define PDIM 7 //7x7 퍼즐의 크기
#define PNUM MDIM*MDIM // 네모칸의 수

//게임 Map 그리기 시작 위치: 필요하다면 출력 위치를 조정할 수 있음
#define MAP_STARTX 15
#define MAP_STARTY 3

//이동 회수 및 소요시간 출력 위치: 필요하다면 출력 위치를 조정할 수 있음
#define SCORE_STARTX 34
#define SCORE_STARTY 36

enum COMMAND { Stop = 0, Left, Right, Down, Up, Non };

class puzzleGame
{
public:

    puzzleGame(); //default 생성자: 클래스 객체 초기화 함수, 필요시에 매개변수를 추가함
    ~puzzleGame();// 소멸자: 클래스 객체의 소멸전에 동적할당의 해제와 같은 객체의 정리를 위해 사용됨
    friend ostream & operator <<(ostream & outputStream, const puzzleGame& gmBot);
                    // gameDrawd 멤버함수 대신에 << 연산자 오버로딩를 통해 사용하시오.
    void puzzleDraw(int puzzle[][PDIM][PDIM], int x, int y, int k) const; // 하나의 퍼즐을 그리는 멤버함수
    void puzzleMove(int moveKey); //퍼즐 이동 함수; 필요시에 매개변수 추가함
    void shuffle(int num, const puzzleGame& gmBot);//퍼즐 맵 초기화; 50회 무작위 이동, 필요시에 인자 추가 할 수 있음
    void getCommand(void);//모든 퍼즐이 제 위치에 있거나 ESC키가 눌러졌으면 정수 0을 반환
                          // 방향키를 눌렀을 때는 해당 키값을 반환(LEFT, RIGHT, DOWN, UP)
                          //필요시에 매개변수 인자를 추가할 수 있음
    int getAction(void) const { return action; }; //메인함수에서 action의 값을 읽는다.
    int solvable(void); //풀 수 있는 퍼즐인지 판별하는 함수
    void setCTime(clock_t ct) { cTime = ct; }; //cTime설정 인라인 함수
    void setSTime(clock_t st) { sTime = st; }; //sTime설정 인라인 함수

private:

    int(*numPat)[PDIM][PDIM];
    int **map;

    int action;
    int x, y; //빈칸의 위치
    int moveNum; //이동회수
    clock_t sTime; //시작 시간
    clock_t cTime; //시작 시간
};

void gotoXY(int x, int y); //콘솔 화면에서 커서를 특정 위치로 이동
int getDirectKey();

int main(void)
{
    puzzleGame gameBot;

    gameBot.shuffle(50, gameBot); //50번 섞는다
    gameBot.setCTime(clock());
    gameBot.setSTime(clock()); //시작하기전 시간 초기화
    cout << gameBot; //게임 화면 출력
    gameBot.getCommand(); //입력, 퍼즐 완성 여부 확인

    while (gameBot.getAction()) //action이 Stop이면 반복문이 종료된다.
    {
        gameBot.puzzleMove(gameBot.getAction()); // 입력된 키에 따라 퍼즐을 움긴다.
        gameBot.setCTime(clock()); //현재 시간을 저장한다.
        cout << gameBot; // 화면출력
        gameBot.getCommand(); //입력, 퍼즐 완성 여부 확인
    }

    system("cls"); //화면을 한번 지운다.
    cout << gameBot; //화면 출력
    gotoXY(SCORE_STARTX + 20, SCORE_STARTY);
    cout << "The game is over!!" << endl << endl << endl; //안내문을 출력한다.
}

ostream & operator <<(ostream & outputStream, const puzzleGame& gmBot)
{ // gameDrawd 멤버함수 대신에 << 연산자 오버로딩를 통해 사용하시오.
    gotoXY(SCORE_STARTX, SCORE_STARTY);
    cout << "  이동 회수: " << gmBot.moveNum << endl;
    gotoXY(SCORE_STARTX, SCORE_STARTY + 1);
    cout.setf(ios::fixed);
    cout.precision(1);
    cout << "  소요 시간: " << (gmBot.cTime - gmBot.sTime) / 1000.0 << "초" << endl;
    cout.unsetf(ios::fixed);
```

```
    cout << "  소요 시간: " << (gmBot.cTime - gmBot.sTime) / 1000.0 << "초" << endl;
    cout.unsetf(ios::fixed);
    //이동 회수 출력, 소요시간 소수점 첫째 자리까지 출력

    gotoXY(MAP_STARTX + 20, MAP_STARTY);
    cout << "   Fifteen Puzzle" << endl; //제목 출력

    for (int i = 0; i< MDIM; i++)
        for (int j = 0; j< MDIM; j++)
            gmBot.puzzleDraw(gmBot.numPat, MAP_STARTX + (j * 15), MAP_STARTY + (i * 8) + 1, gmBot.map[i][j]);
    //puzzleDraw 함수 호출해 퍼즐 출력.

    return outputStream;
}

puzzleGame::puzzleGame()
{

    int num = 1, temp;
    ifstream fin;

    numPat = new int[PNUM][PDIM][PDIM];
    fin.open("..//NumberPattern.txt"); // "NumberPattern.txt" 파일이 프로젝트 폴더에 위치해야 함
    if (fin.fail())
    {
        cout << "숫자 패턴 파일을 열 수 없습니다!\n";
        exit(1);
    }

    for (int p = 0; p < PNUM; p++)
        for (int i = 0; i< PDIM; i++)
            for (int j = 0; j< PDIM; j++)
            {
                fin >> temp;
                numPat[p][i][j] = temp;
            }

    map = new int*[MDIM]; // 이차원 포인터를 동적할당 한다.

    for (int i = 0; i < MDIM; i++)
        map[i] = new int[MDIM]; // 이차원 포인터를 동적할당한다.

    for (int i = 0; i < MDIM; i++)
        for (int j = 0; j < MDIM; j++)
            map[i][j] = i * 4 + j + 1;
    //퍼즐 맵 동적할당 및 값 할당
    map[MDIM - 1][MDIM - 1] = 0;
    x = MDIM - 1; y = MDIM - 1;
    // 맵의 값을 1,2,3,~ ,MDIM*MDIM-1,0의 순서로 초기화 시키고 빈칸의 위치를 마지막으로 초기화시킨다.
    sTime = 0; //시작시간을 초기화 시킨다.
    moveNum = 0; //이동 회수를 초기화시킨다.

                //게임 초기화 코드 작성
}

puzzleGame::~puzzleGame()
{
    for (int i = 0; i < MDIM; i++)
        delete[] map[i]; //배열 포인터를 초기화한다.

    delete[] map; // 배열 포인터를 초기화한다.
    delete[] numPat; //포인터 배열을 초기화 한다.
}

void puzzleGame::puzzleDraw(int puzzle[][PDIM][PDIM], int x, int y, int k) const
{ //7*7의 숫자 형태와, 입력할 위치, 입력할 숫자를 입력받는다.
    for (int i = 0; i< PDIM; i++)
    {
        gotoXY(x, y + i);
        for (int j = 0; j< PDIM; j++)
        {
            if (puzzle[k][i][j] == 0)
                cout << "  ";
            else if (puzzle[k][i][j] == 1)
                cout << "■";
            else
                cout << "□";
        }
    } //숫자 형태의 모양에 따라 기호를 다르게 출력한다.
}

void puzzleGame::puzzleMove(int moveKey)
{
    int temp = 0;

    switch (moveKey) //getcommand를 통해 입력받은 키보드 값으로 퍼즐을 움직인다.
    {
    case Left:
        if (x >= 1) { //예외 처리를 통해 빈칸의 위치가 판을 벗어나지 않도록 한다.
            temp = map[y][x];
            map[y][x] = map[y][x - 1];
            map[y][x - 1] = temp;
            x--; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
        }
        break;
```

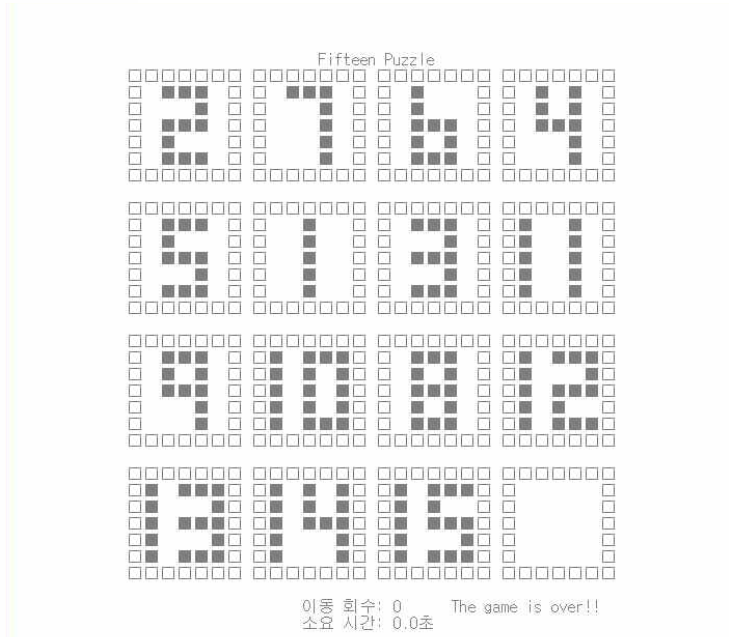
```
187             }
188             break;
189
190         case Right:
191             if (x <= 2) {
192                 temp = map[y][x];
193                 map[y][x] = map[y][x + 1];
194                 map[y][x + 1] = temp;
195                 x++; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
196             }
197             break;
198
199         case Up:
200             if (y >= 1) {
201                 temp = map[y][x];
202                 map[y][x] = map[y - 1][x];
203                 map[y - 1][x] = temp;
204                 y--; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
205             }
206             break;
207
208         case Down:
209             if (y <= 2) {
210                 temp = map[y][x];
211                 map[y][x] = map[y + 1][x];
212                 map[y + 1][x] = temp;
213                 y++; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
214             }
215             break;
216
217         default:
218             break; //아무것도 아니면 그냥 switch를 탈출한다.
219         }
220     }
221
222     void puzzleGame::getCommand(void)
223     {
224         bool Flag = FALSE; //퍼즐이 맞추어졌는지 확인한다.
225
226         for(int i=0;i<MDIM;i++)
227             for (int j = 0; j < MDIM; j++) {
228                 if ((map[i][j] != i * 4 + j + 1) && (i*j < (MDIM-1)*(MDIM-1))) { Flag = TRUE; }
229             } //중간에 틀린 부분이 발생하면 Flag가 TRUE로 설정된다.
230         if (Flag == FALSE)
231             action = Stop; //퍼즐이 다 맞으면 action 을 Stop으로 한다.
232
233         if (action != Stop) //퍼즐이 맞추어지지 않은 경우 키보드로 키를 입력받는다.
234             switch (getDirectKey()) { //getDirectKey 함수를 통해 키를 입력받고 명령어를 디코딩해 action에 저장한다.
235             case ESC:
236                 action = Stop;
237                 moveNum = 0;
238                 sTime = cTime = 0; //시간과 이동횟수를 초기화한다.
239                 break;
240             case LEFT:
241                 action = Left;
242                 break;
243             case RIGHT:
244                 action = Right;
245                 break;
246             case DOWN:
247                 action = Down;
248                 break;
249             case UP:
250                 action = Up;
251                 break;
252             default:
253                 action = Non;
254                 break; // 방향키나 ESC키말고 다른키가 입력되면 NON을 리턴한다.
255             }
256     }
257
258     void puzzleGame::shuffle(int num, const puzzleGame& gmBot)
259     {
260         int numcheck[MDIM*MDIM]; //배열의 값이 중복되었는지 체크하기위한 맵과 동일한 크기의 배열을 생성한다.
261         int random = 0; //생성된 난수를 저장해 놓는 변수이다.
262
263         srand(time(NULL)); //난수의 시드 값을 바꾸어 줌으로써 난수 생성을 한다.
264
265         for (int k = 0; k < num; k++)
266         {
267             while (1) {
268                 for (int i = 0; i < MDIM*MDIM; i++)
269                     numcheck[i] = 1; //난수 설정하기전에 중복체크 배열을 사용되기 전 상태인 '1'로 초기화 시킨다.
270
271                 for (int i = 0; i < MDIM; i++)
272                     for (int j = 0; j < MDIM; j++)
273                         while (1) {
274                             if ((numcheck[random = (rand() % (MDIM*MDIM))]) != 0) {
275                                 if (random == 0) { //만약 난수의 값이 빈칸의 좌표인 0이라면 그의 배열 좌표를 저장한다.
276                                     x = j; //빈칸의 위치를 저장하기 위해 멤버 변수 x,y에 2차원 배열의 좌표를 저장한다.
277                                     y = i;
278                                 }
279                                 map[i][j] = random;
280                                 numcheck[random] = 0;
281                                 break;
```

```
                                numcheck[random] = 0;
                                break;
                            } //생성된 난수가 사용되지 않은 난수이면 사용 표시를 하고 배열에 저장하며 루프를 탈출한다.
                        }     // 난수가 사용 전이면 numcheck에 1의 형태로, 후면 0의 형태로 저장된다.

                if (solvable() == 1)//퍼즐을 풀지 못하는 경우로 섞일 수 있다. 풀 수 있는지 판별하여
                    break;                //풀 수 있으면 1을 리턴하며 풀지못하면 0을 리턴하고 다시 섞는다.
            }
            cTime = sTime = 0; //섞을 때는 시작 시간을 계속 초기화 시켜줌으로써 시간이 0으로 보이도록 한다.
            cout << gmBot;
            Sleep(100); //맵 변경이 보이도록 대기시간을 준다.
        }
    }

int puzzleGame::solvable(void) //무작위로 배열된 퍼즐이 풀수있는지 판별하는 함수
{
    int inv_count = 0, k = 0, temp, t_arr[MDIM*MDIM]; //반전된 숫자 배열 세는 변수 등을 선언

    if (x != MDIM - 1 || y != MDIM - 1) //먼저 빈칸의 위치를 맨 아래 맨 오른쪽으로 이동시킨다.
    {
        temp = map[MDIM - 1][MDIM - 1];
        map[MDIM - 1][MDIM - 1] = 0;
        map[y][x] = temp;
        y = MDIM - 1;
        x = MDIM - 1;
    }

    for (int i = 0; i < MDIM; i++)
        for (int j = 0; j < MDIM; j++)
            t_arr[i * 4 + j] = map[i][j];
    //배열의 숫자들을 비교하기 위해 퍼즐 크기의 일차원 배열에 복사한다.

    for (int j = 0; j < MDIM * MDIM - 1; j++)
        if (t_arr[j + 1] > t_arr[j])
            inv_count++;
    //일차원 배열들의 값을 순차적으로 비교하며 반전된 수의 수를 센다.

    if (inv_count % 2 == 1) //만약 반전된 수의 수가 홀수 개이면 풀 수 없는 퍼즐이며 0을 리턴한다.
        return 0;
    else
        return 1; //짝수 개이면 풀 수 있는 퍼즐이며 1을 리턴한다.
}

 //콘솔 화면에서 커서를 특정 위치로 이동
void gotoXY(int x, int y)
{
    COORD Pos = { x, y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}

int getDirectKey()
{
    char key = Non;
    while (1) { //키보드가 눌릴때만 루프를 탈출한다.
        if (_kbhit() != 0) //키보드를 눌렀는지 확인함
        {
            //특수 키를 눌렀을 때 버퍼에 2Byte가 발생함, 첫번째 값은 224값을 발생하고 두번째 값은 특수키에 따라 다름
            //특수 키를 확인하기 위해 2번의 getch()함수를 호출해야 함
            key = _getch();
            if (key == 224) key = _getch();
            break;
        }
    }
    return key;
}
```

<게임 시작 단계>



<게임 종료 단계>



<게임 도중 단계>