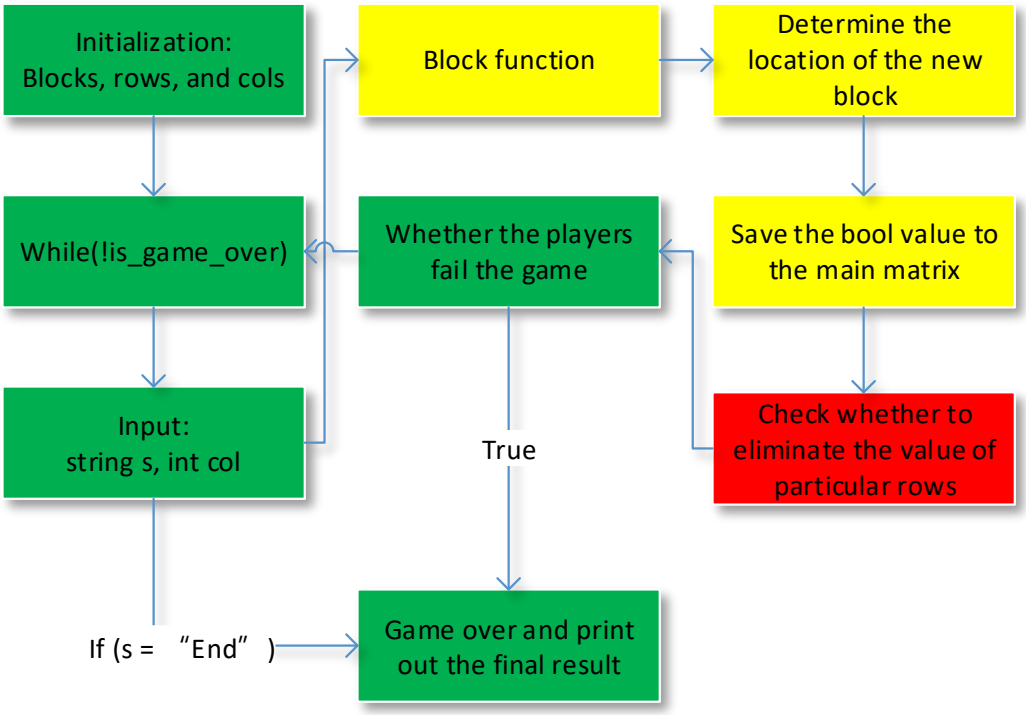
1 Project Description

1.1 Program Flow Chart



1.2 Detailed Description

設計思路：

一開始在寫 project 的時候發現，方塊的形狀都是固定的，於是想要用二為矩陣去儲存方塊的相關資料，例如：

bool T1[2][3] = {1, 1, 1, 0, 1, 0};

但是當我要寫入 function 的時候，會有很大的麻煩，因為我們無法直接將一個固定的二維矩陣的 pointer to pointer 直接傳入至一個 function 裡（除非一開始是用 new 去建立二維矩陣），我以以下的 code 為例子：

```cpp
#include <iostream>
using namespace std;

void print(int *matrix) {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            cout << matrix[3*i+j] << ' ';
        }
        cout << endl;
    }
}

int main() {
    int matrix[3][3];
    int value = 1;
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            matrix[i][j] = value++;
        }
    }
    print(*matrix);
    return 0;
}
```

從上面的 example 中會發現在傳入 print 的是 matrix 的 row 的 address，因此在 function 裡 matrix 變為一維矩陣來使用，那還不如一開始全部都用一維矩陣來執行，反而更有效率且方便。於是我決定使用一維矩陣並以 pointer 來進行演算。

演算過程：

首先先定義變數以及方塊形狀（方塊的布林值取的順序是由上到下由左到右）：

```
    bool T1[6] = {1, 1, 1, 0, 1, 0};
    bool T2[6] = {0, 1, 1, 1, 0, 1};
    bool T3[6] = {0, 1, 0, 1, 1, 1};
    bool T4[6] = {1, 0, 1, 1, 1, 0};
    bool L1[6] = {1, 0, 1, 0, 1, 1};
    bool L2[6] = {1, 1, 1, 1, 0, 0};
    bool L3[6] = {1, 1, 0, 1, 0, 1};
    bool L4[6] = {0, 0, 1, 1, 1, 1};
    bool J1[6] = {0, 1, 0, 1, 1, 1};
    bool J2[6] = {1, 0, 0, 1, 1, 1};
    bool J3[6] = {1, 1, 1, 0, 1, 0};
    bool J4[6] = {1, 1, 1, 0, 0, 1};
    bool S1[6] = {0, 1, 1, 1, 1, 0};
    bool S2[6] = {1, 0, 1, 1, 0, 1};
    bool Z1[6] = {1, 1, 0, 0, 1, 1};
    bool Z2[6] = {0, 1, 1, 1, 1, 0};
    bool I1[4] = {1, 1, 1, 1};
    bool I2[4] = {1, 1, 1, 1};
    bool O[4] = {1, 1, 1, 1};
    bool is_game_over = 0;
    int rows, cols;
    string s;
```

然後開啟檔案 tetris.data，並決定玩家想要的長度和寬度：

```
ifstream fin("tetris.data");
fin >> rows >> cols;
```

接著判斷玩家所輸入的內容是否合法，接著增加 row 的數量，主要是給予方塊掉下來的空間：

```
    if (rows > 40 || cols > 15) {
        throw "Error! The size of row or col is too large!";
    }
    if (rows < 1 || cols < 1) {
        throw "Error! The size of row or col is too small!";
    }
    rows += 4;
```

然後建立一個一維 matrix 並且 initialize。

```
    bool *matrix = new bool[rows * cols];
    for (int i = 0; i < rows * cols; ++i) {
        matrix[i] = 0;
    }
```

接下來我們就可以開始輸入我們想丟入的方塊了。我們的 input 主要有兩個，s 和 col，其中 s 代表我們所輸入的字串，col 代表我們想讓方塊在哪一列掉落。由於 switch 無法接受字串的判斷，因此這裡全部由 if...else 來控制。我們所輸入的值經過判斷後，會輸入至一個叫做 block 的 function，其中 function 所輸入的變數有 7 個：

```
void block(bool *matrix, int matrix_rows, int matrix_cols,
bool *block, int block_rows, int block_cols, int col);
```

在 block 中，先定義幾個重要變數，並且判斷玩家輸入的內容是否合法（方塊的位置是否超出範圍），其中 p 代表一開始方塊的左上角的 address，row 代表一開始方塊的最底部在第幾行：

```
    bool *p = matrix + col - 1;
    int row = block_rows - 1;

    if (col + block_cols - 1 > matrix_cols) {
        throw "Error input!";
    }
```

接著就是判斷方塊掉落的行數，由於 row 不能超出 matrix 的範圍，因此該迴圈由 `while (row < matrix_rows)`控制。在迴圈中定義一些非常重要的變數：q 代表同一列中 p 下一行的 address，r 代表該方塊的 address。第一個雙重 for loop 是用來 check 該方塊若再往下一格，是否與其他方塊重疊，若有重疊或是已經到底，則進入第二個雙重 for loop，注意這裡的 q 等於 p，r 變回原本的 block。

第二個雙重 for loop 是用來記錄該方塊最後掉落的地方，`*q = (*r) ? 1 : *q;`這條式子就是將 block 的值紀錄再 matrix 上，要注意不能直接讓 q = p，否則會讓本來已經為 1 的值變為 0。記錄完後，必須跳出迴圈 while，直接 break。

```
    while (row < matrix_rows) {
        bool *q = p + matrix_cols;
        bool *r = block;
        bool check = 0;
        for (int i = 0; i < block_rows && row < matrix_rows; ++i) {
            for (int j = 0; j < block_cols; ++j) {
                if (*q && *r) {
                    check = 1;
                }
                ++r; ++q;
            }
            q = q + matrix_cols - block_cols;
        }
        q = p; r = block;
        if (check || row == matrix_rows - 1) {
            for (int i = 0; i < block_rows; ++i) {
                for (int j = 0; j < block_cols; ++j) {
                    *q = (*r) ? 1 : *q;
                    ++r; ++q;
                }
                q = q + matrix_cols - block_cols;
            }
            break;
        }
        p += matrix_cols; ++row;
    }
```

接著，既然位置已經記錄好了，那就要判斷該行能否進行消除，輸入至 eliminate 的變數只要有三個就好，因為不必再對 matrix 進行增加方塊的動作：

```
eliminate(matrix, matrix_rows, matrix_cols);
```

首先先再函數裡定義幾個重要的變數：p 為 pointer，作為預備用，is_full 預設為 1 ，為了讓後面能夠進入 while 檢查，eliminate_row 紀錄要被削除的行數，其預設值為 0 。

```
    bool *p;
    bool is_full = 1;
    int eliminate_row = 0;
```

進入 while 迴圈時，第一個 for 迴圈不會被執行，因為 eliminate_row 的預設值為 0 。然後讓 p 為 matrix，讓 p 能夠從頭到尾檢查哪一行全部都為 1 ，若其中該行有一個 element 為 0 ，則 is_full 就會一直為零，直到進入下一行為止。若檢查到有一行全部為 1 時，eliminate_row 就會記錄第幾行，然後再執行一次 while loop，這時候第一個 for loop 就會執行，*p = *(p - matrix_cols);這條式子主要是位移被削除的那行以上所有的 element 往下移動一行。移動完後，就會再次檢查，直到沒有能夠被削除的 element。

```
while (is_full) {
    for (int i = eliminate_row; i > 0; --i) {
        p = matrix + matrix_cols * i;
        for (int j = 0; j < matrix_cols; ++j) {
            *p = *(p - matrix_cols);
            ++p;
        }
    }
    p = matrix;
    for (int i = 0; i < matrix_rows; ++i) {
        is_full = 1;
        for (int j = 0; j < matrix_cols; ++j) {
            is_full *= *p;
            ++p;
        }
        if (is_full) {
            eliminate_row = i;
            break;
        }
    }
}
```

回到原本的 main，的 while loop 裡，後面有一個判斷式為：

```
    for (int i = 0; i < cols * 4; ++i)
        if (matrix[i])
            is_game_over = 1;
```

其主要在於檢查 matrix 上面四行是否有布林值為 1 的 element，若有，則結束遊戲。

最後關閉 tetris.data，並且建立 tetris.output，然後印出最後的結

果，再關閉 tetris.output：

```
    fin.close();
    ofstream fout("tetris.output");
    for (int i = 4; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            fout << matrix[i * cols + j];
        }
        fout << endl;
    }
    fout.close();
```

到這裡該 project 已結束。

2 Test case Design

2.1 Detailed Description of the Test Case

Test case:

12 9	O 8	T1 1	L2 2
T3 1	O 8	O 4	I2 1
T2 3	T2 5	S2 5	T2 5
O 5	T2 5	L1 7	L1 7
J2 7	I1 7	T2 8	L3 8
O 8	Z1 1	O 1	I2 2
I1 1	Z1 1	L3 2	L2 1
O 2	I1 4	J3 4	I2 4
T2 4	O 8	T1 7	J1 8
O 6	Z1 2	J3 6	End
L4 2	O 7	O 8	
J4 6	L2 1	Z2 1	
O 1	L3 8	L2 3	
I2 6	Z2 4	O 6	
Z2 5	T1 6	L3 4	
Z2 3	O 1	T1 2	
S2 4	L2 4	I1 1	
O 8	J4 7	S1 2	
I1 7	I1 3	T2 6	
O 1	L3 4	I1 5	
Z1 1	T4 1	O 8	
T2 5	I2 6	O 8	