Data Structure

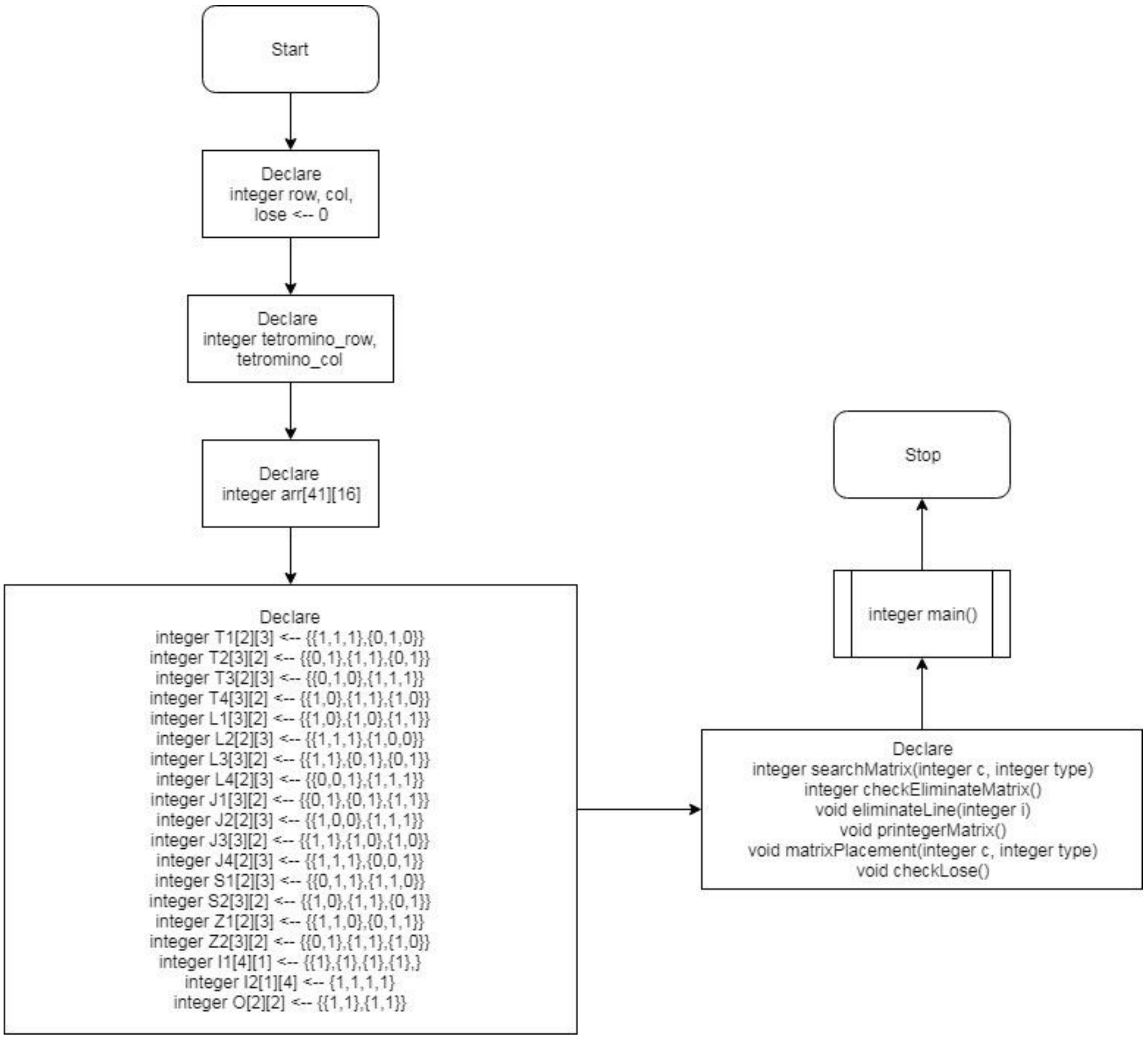
Project 1 -- **Tetris**

學號:107062361　姓名:許珉濠

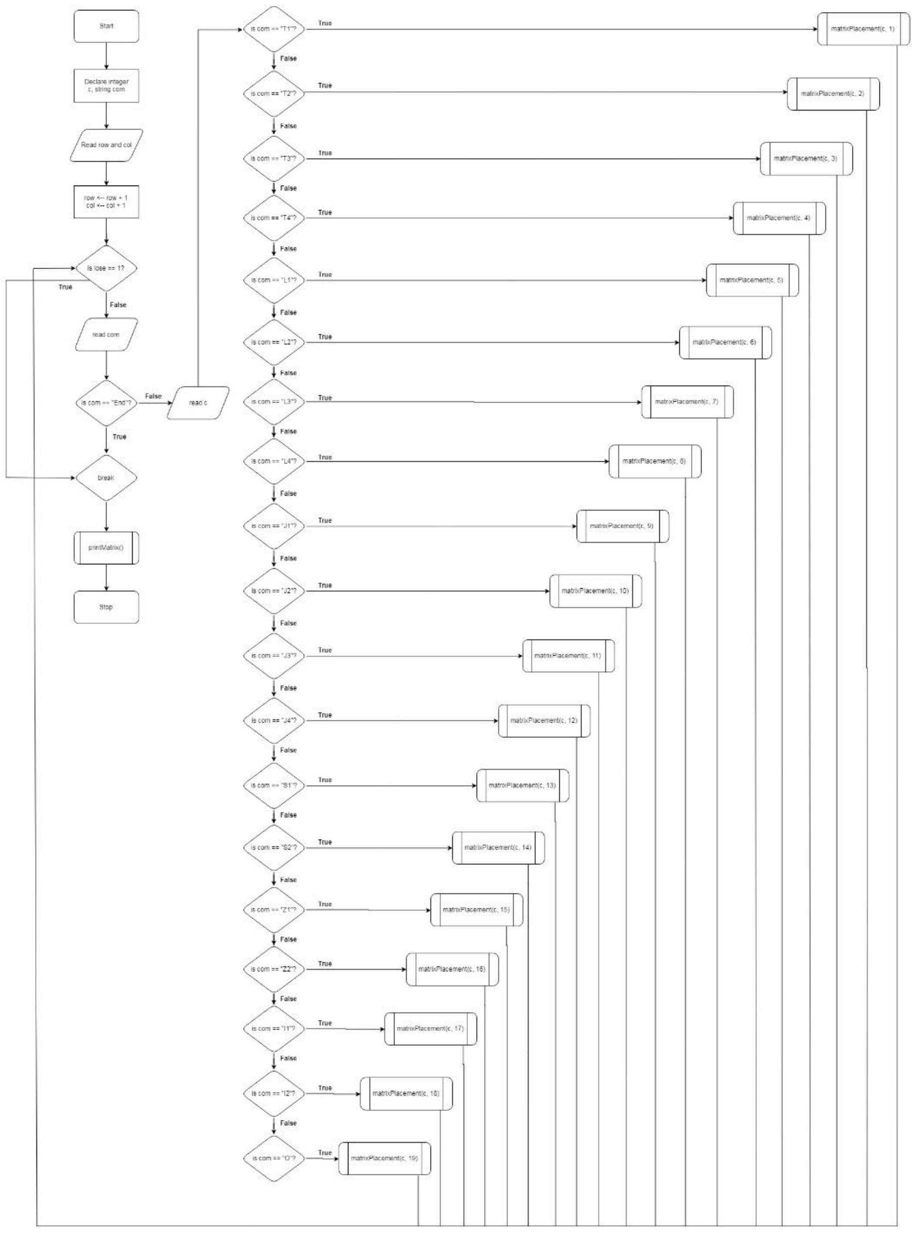
1. Project Description

1.1. Program Flow Chart

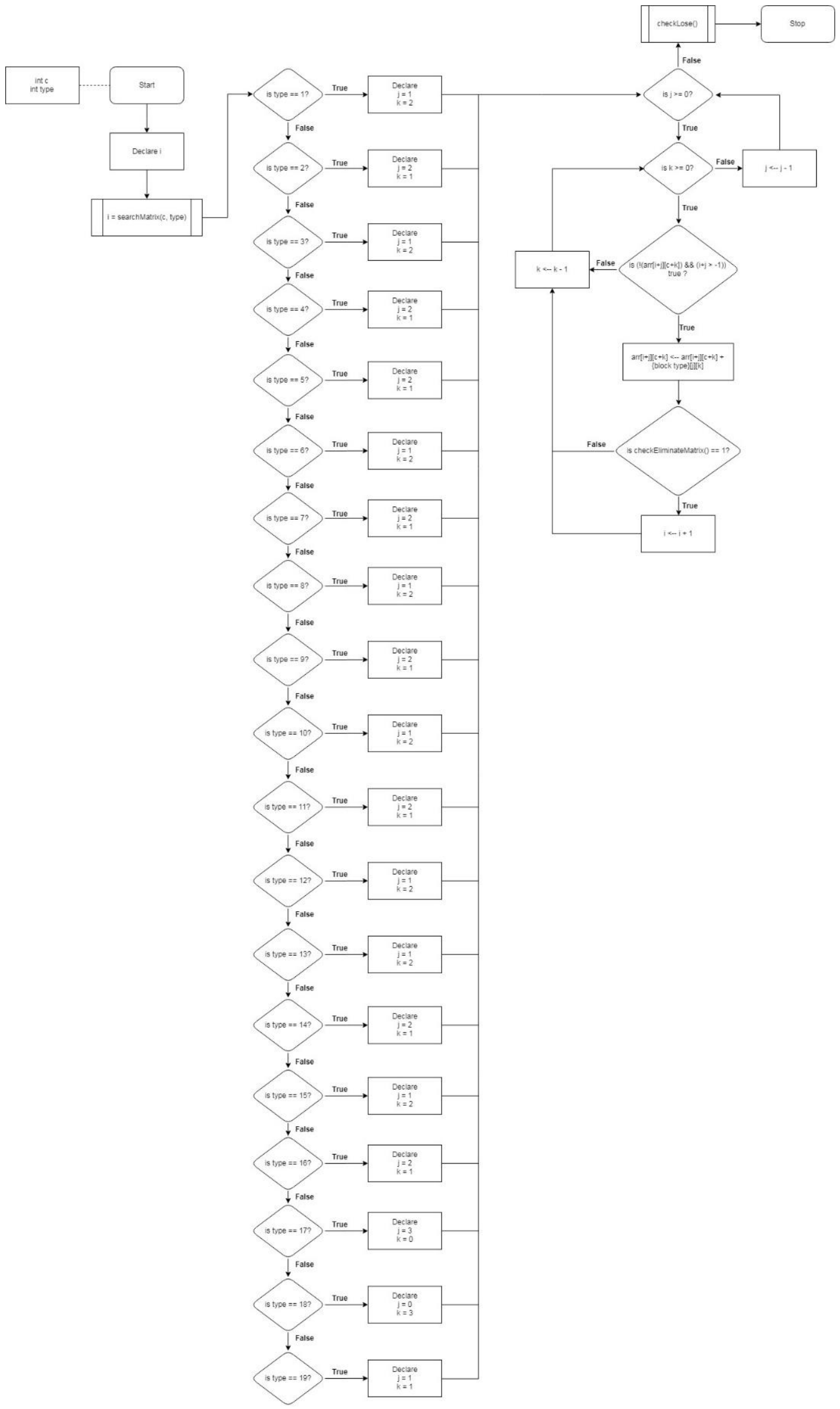
- **Overall Process Flow Chart**


Start
Declare
integer row, col,
lose <-- 0
Declare
integer tetromino_row,
tetromino_col
Declare
integer arr[41][16]
Declare
integer T1[2][3] <-- {{1,1,1},{0,1,0}}
integer T2[3][2] <-- {{0,1},{1,1},{0,1}}
integer T3[2][3] <-- {{0,1,0},{1,1,1}}
integer T4[3][2] <-- {{1,0},{1,1},{1,0}}
integer L1[3][2] <-- {{1,0},{1,0},{1,1}}
integer L2[2][3] <-- {{1,1,1},{1,0,0}}
integer L3[3][2] <-- {{1,1},{0,1},{0,1}}
integer L4[2][3] <-- {{0,0,1},{1,1,1}}
integer J1[3][2] <-- {{0,1},{0,1},{1,1}}
integer J2[2][3] <-- {{1,0,0},{1,1,1}}
integer J3[3][2] <-- {{1,1},{1,0},{1,0}}
integer J4[2][3] <-- {{1,1,1},{0,0,1}}
integer S1[2][3] <-- {{0,1,1},{1,1,0}}
integer S2[3][2] <-- {{1,0},{1,1},{0,1}}
integer Z1[2][3] <-- {{1,1,0},{0,1,1}}
integer Z2[3][2] <-- {{0,1},{1,1},{1,0}}
integer I1[4][1] <-- {{1},{1},{1},{1},}
integer I2[1][4] <-- {1,1,1,1}
integer O[2][2] <-- {{1,1},{1,1}}
Declare
integer searchMatrix(integer c, integer type)
integer checkEliminateMatrix()
void eliminateLine(integer i)
void printegerMatrix()
void matrixPlacement(integer c, integer type)
void checkLose()
integer main()
Stop

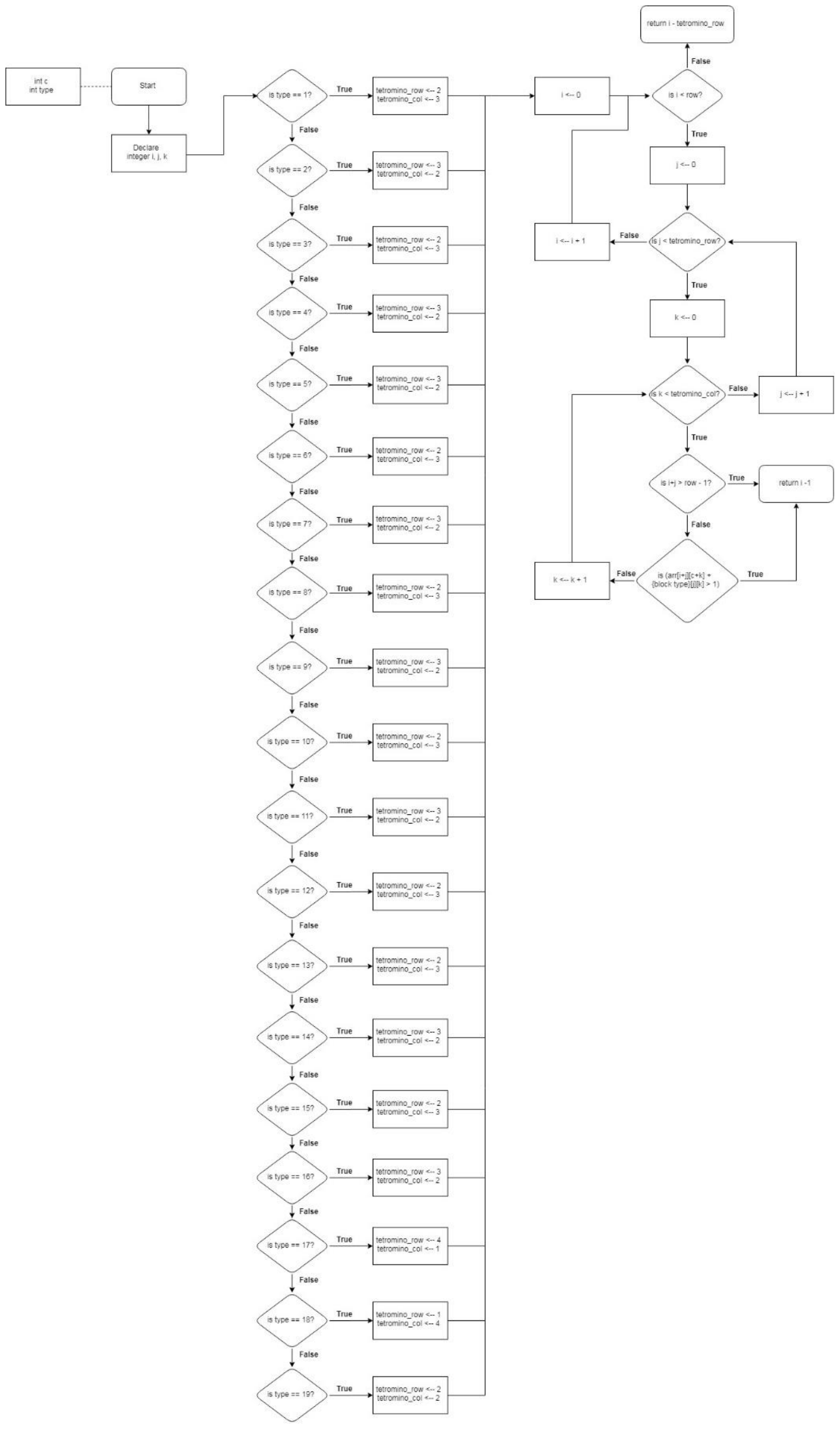
- **int main() Flow Chart**



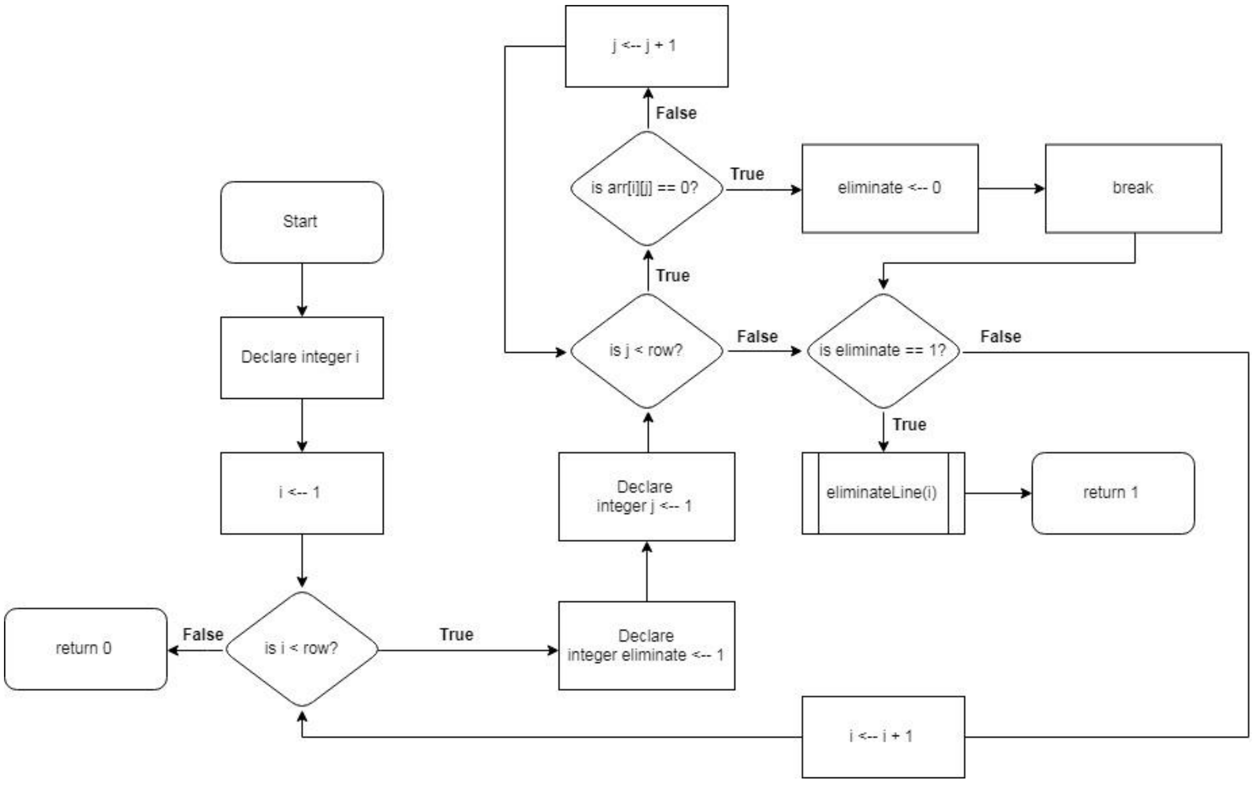
- **matrixPlacement() Flow Chart**



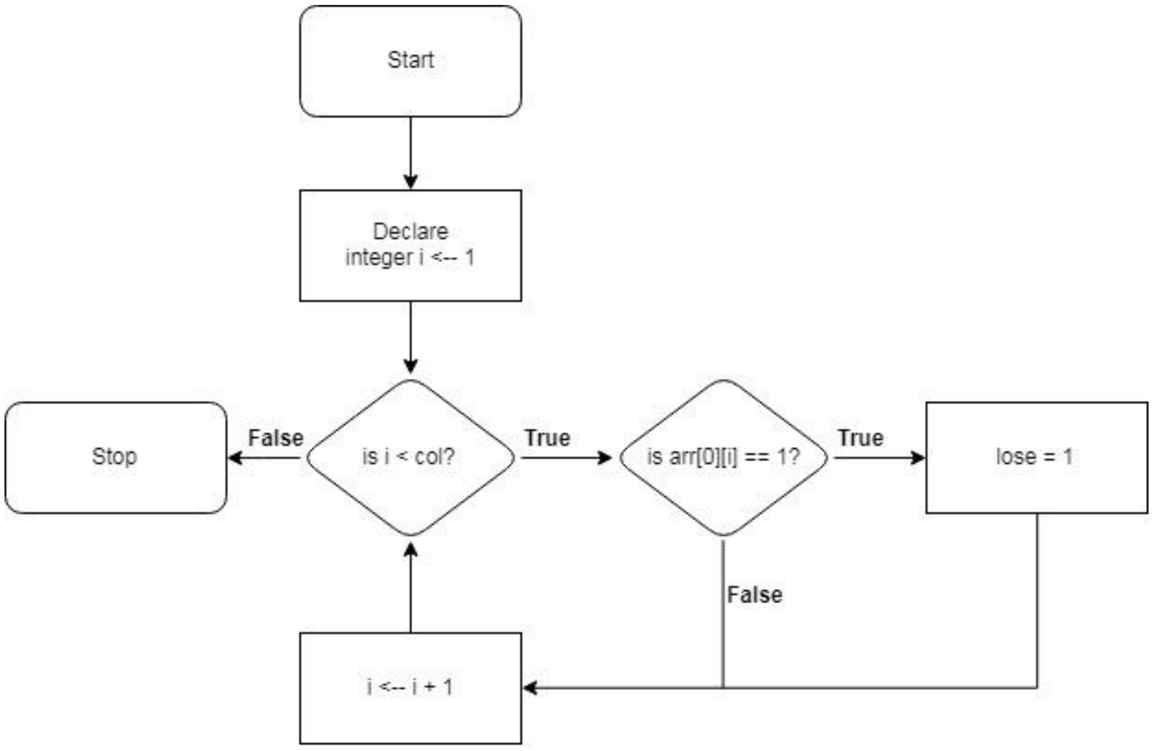
- **searchMatrix Flow Chart**



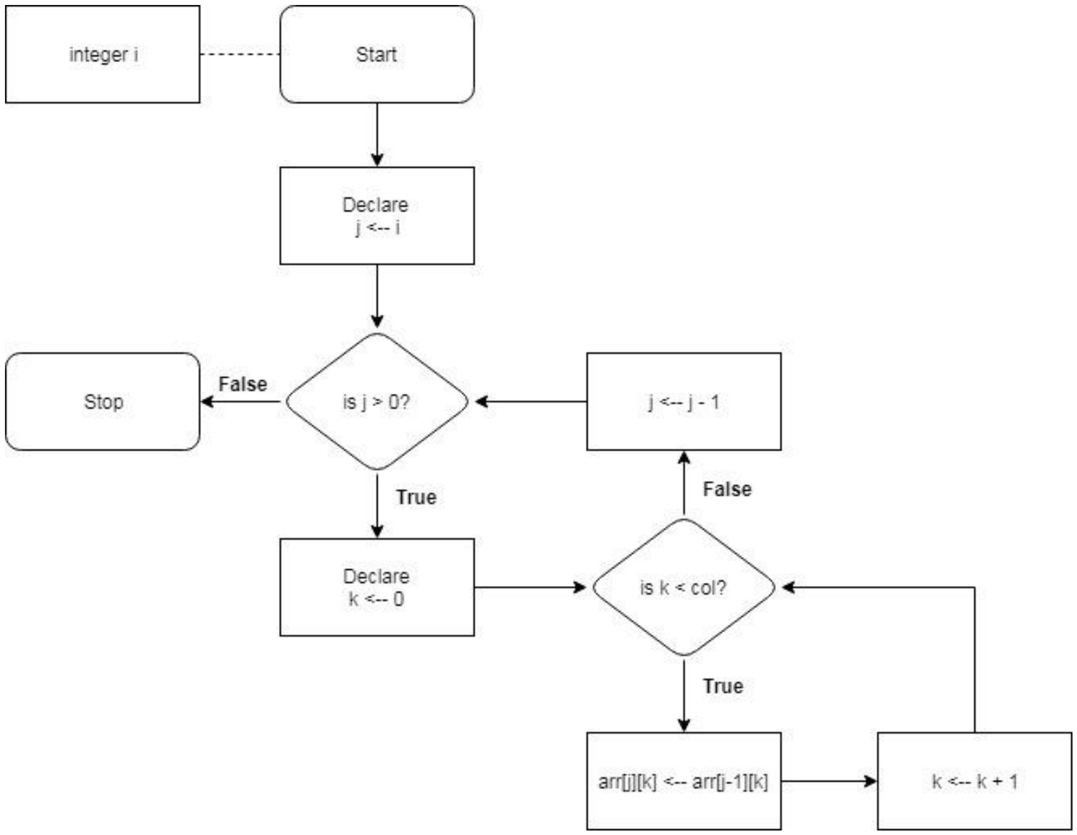
- **checkEliminateMatrix Flow Chart**



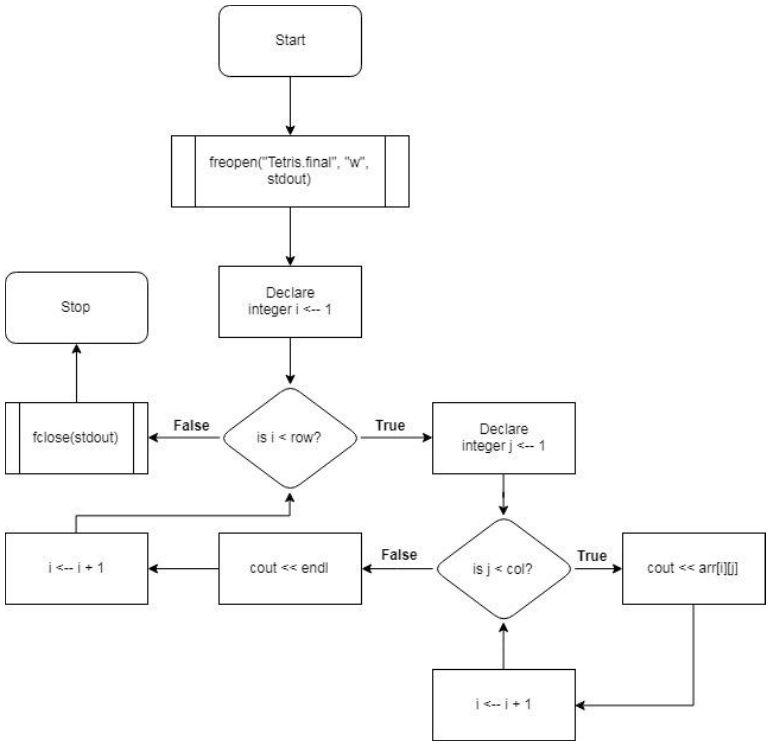
- **checkLose Flow Chart**



- **eliminateLine Flow Chart**



- **printMatrix Flow Chart**



1.2. Detailed Description

The declared variable in overall flowchart is global variable that will be used in this program. The purpose of declaring global variable is to avoid passing variable to function which sometimes can provoke mistake in writing code. First, the integer **row** and **col** is the variable that will be used as the size of row and column of matrix. The integer **lose** is a flag to determine to break the loop in main function. About **tetromino_row** and **tetromino_col**, they will be the indicator size of tetris block which will be used in **searchMatrix** function. The arr[41][16] is a matrix array with size 41x16 since the index is started from 1 and the maximum size requirement is 40x15. There are also 19 type of tetris block that is declared with the determined size in the given project description. Afterwards, the main function will be started. The program will stop after main function end.

The main function flowchart is started with declaring integer **c** and string **com** which will be used as input. In this case, **c** means the starting column the lower left corner of the corresponding block to fall on. On the other hand, **com** means command that will end the program if **com** equal to "**End**". Otherwise, it will become the selector of tetris block which placement depends on integer **c**.

Subsequently, read **row** and **col** is the process of input row and colum which values will be stored in **row** and **col**. The row and col is incremented by one due to row and column is started from index 1.

The remaining program will run inside infinite while loop. It is started with if statement "is **lose** == 1" which **lose** indicate as flag to break the loop if true. On the contrary, the program will start read string **com.** If "is **com** == "End" is true, then the program will stop by breaking the loop. Assuming that it is false, the input must be name of one tetris block.

The next step is read **c** value which will be inserted into **matrixPlacement** function which parameter is integer **c** and integer **type**. The integer **type** means the type of tetris block that will fall off according to **type** value. In the following table, it will show the specified **type** value that refers the tetris block.

Type	Tetris Block	Type	Tetris Block	Type	Tetris Block	Type	Tetris Block
1	T1	6	L2	11	J3	16	Z2
2	T2	7	L3	12	J4	17	I1
3	T3	8	L4	13	S1	18	I2
4	T4	9	J1	14	S2	19	O
5	L1	10	J2	15	Z1		

The process of reading c will keep continue unless the **com** is “End”. Before the program ends, it will call **printMatrix** function to store the current condition of matrix.

In the **matrixPlacement** function, it will begin with integer **i** that stores the return value of **searchMatrix**. The interger **i** is the row of matrix where upper left corner of tetris block occupies based on tetris block type. The way of placing tetris block is through nested for loop. It will check whether the place is suitable to be placed by statement “**if(!(arr[i+j][c+k]) && (i+j > -1))**”. Suppose the place is able to placed, the program will start store the value in the matrix. A function called **checkEliminateMatrix** will be called in order to check the horizontal line. If it has no gaps, the horizontal line blocks will be discarded and the matrix blocks will go down 1 row. The integer **i** is incremented since matrix blocks go down 1 row. Soon, **checkLose** function is called to ensure the game has ended.

In respects of **searchMatrix** function, it is similar to **matrixPlacement**. It will declare integer **i**, **j**, and **k** that will be used in nested for loop. Since the function parameter is integer **c** and **type**. The program will run based on type of tetris block. In the nested loop, the suitable row of matrix placement is checked from index 0. An if statement “**i+j > row – 1**” is used to prevent the the row of matrix placement exceed the row size. Consider that the current row of matrix placement is already occupied, the previous row will be returned. Contrarily, the “**row – tetromino**_row” size is returned.

The method to check the elimination of horizontal line in **checkEliminateMatrix()** is applying nested loop that will check if each row has no gaps. On the assumption that the horizontal line has no gaps, it will start eliminate the line by calling **eliminateLine** function. The goal of return value 0 or 1 is to inform that the **matrixPlacement** function has to increment **i**.

The way to determine the end of game is quite simple. With for loop, the thing that needs to be checked is value of each column of row 0. The reason that row 0 is chosen since it is out of matrix scope. If one of column value in row 0 is 1, then assign **lose** with 1. Thus, this is the process that is happened in **checkLose** function.

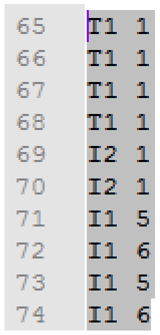
Pertaining to **eliminateLine** function, the things that needs to do is operate nested loop to replace the row **i** with row **i-1** until row 1. For **printMatrix** function, the **freopen** function is writing all printed output into “**Tetris.final**” as soon as the function is called. In this case, nested for loop is used to print the value of matrix. The function ends with “fclose(stdout)” that means stop write the printed output.

2. Test Case Design

2.1. Detailed Description of the Test Case

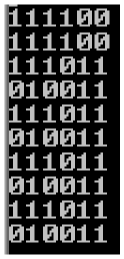
The test case I designed is having matrix 10x6 with 988 tetris block that will fall down. In the test case, I use all the 19 types of tetris block that will fall down in the different column between index 1 to 6. The purpose of using all 19 type of tetris block is to strengthen my test case. By virtue of the program that is designed by some students might have bug in some types of block.

In addition, I also add some test case that will surpass the top of matrix. However, the games does not end due to one rule. If the surpasses block can return back into the scope of matrix after line elimination. The operation of game will keep continue. An illustration of the problem in my test case is in part of following picture.



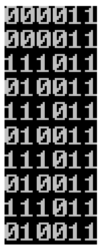
```
T1 1
T1 1
T1 1
T1 1
I2 1
I2 1
I1 5
I1 6
I1 5
I1 6
```

It will give the output in below picture.



```
111100
111100
111011
010011
111011
010011
111011
010011
111011
010011
```

After I add "**L1 5**" and "**L3 5**", the program will keep continue since row 1 and 2 has been eliminated. The top part from **L1** and **L3** tetris block that is not eliminated will be pushed down to row 1 and 2. The result of elimination can be seen beneath.



```
000011
000011
111011
010011
111011
010011
111011
010011
111011
010011
```

Actually the fall down tetris block that I designed is around 100, but I repeat the test case by 7 or 8 times with purpose of testing the execution time of the program. For the other reason that I multiple the test case is the students that cannot pass my test case might feel lazy to debug it in the code revision. Thus, this is my idea of design the test case.