UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS132 - Imperative Programming

Practical 4 Specifications

Release Date: 10-03-2025 at 06:00

Due Date: 14-03-2025 at 23:59

Late Deadline: 16-03-2025 at 23:59

Total Marks: 105

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you begin coding.***
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 48 hours after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

In this practical, you will need to convert a series of flowcharts to boolean expressions as was done in Tutorial 4. You do not need to use any selection or repetition statements for this practical. You are only required to implement the `TicTacToeHelper.cpp` and `TicTacToeHelper.h` files. Note that you have only been provided with provided files. You will need to create the `TicTacToeHelper.h` and `TicTacToeHelper.cpp` files yourself.

3 Background

Tic-Tac-Toe is a simple yet strategic two-player game played on a 3x3 grid. The game is also known as "Noughts and Crosses" in some regions. The objective is to form a straight line of three of one's own symbols, either **X** or **O**, before the opponent does, but for this practical, you will use **1** and **0**.

3.1 Game Rules

The rules of Tic-Tac-Toe are as follows:

1. The game is played on a 3x3 grid.
2. Two players take turns marking a cell with their respective symbol: one plays **1**, the other plays **0**.
3. A player wins if they place three of their symbols in a row, column, or diagonal.
4. If all nine cells are filled without a winner, the game results in a draw.

3.2 Strategy and Winning Tactics

Although the game is simple, players can use strategy to increase their chances of winning:

- The best opening move is often to place an **1** or **0** in the centre, as it provides the most opportunities for forming a line.
- If the centre is taken, a corner is the next best move.
- Blocking the opponent's potential winning moves is crucial.
- If played optimally by both players, the game will always end in a draw.

Tic-Tac-Toe is a fundamental game that introduces players to strategic thinking and pattern recognition. Despite its simplicity, it serves as a basis for more complex games and artificial intelligence studies.

3.3 Terminology

For this practical, the board is represented as a string consisting out of 9 characters. Figure 1 illustrates the mapping between the position of the characters in the string and cells in the board.

0	1	2
3	4	5
6	7	8

Figure 1: Mapping of positions of characters in the string and cells in the board

An **empty board** is represented using the following string (9 space characters[1]):

```
"         "
```

and corresponds to the board in Figure 2.

[1]Space characters are thus used to indicate empty cells.

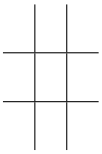


Figure 2: An empty board

A **partially complete** board is a board where not all of the cells have been filled in. The following string represents a partially filled board:

```
"1    0    1  "
```

and corresponds to the board in Figure 3.

1		
	0	
	1	

Figure 3: A partially completed board

A **fully complete** board is a board where all of the cells have been filled in. The following string represents a partially filled board:

```
"101010101"
```

and corresponds to the board in Figure 4.

1	0	1
0	1	0
1	0	1

Figure 4: A fully completed board

A partially and fully complete rows and columns follow the same idea as the board. Consider the following string:

```
" 0   010  1  "
```

which corresponds to the board in Figure 5.

	0	
0	1	0
	1	

Figure 5: Board used to demonstrate fully and partially complete rows and columns

In this example, row 1 is complete, and rows 0 and 2 are partially complete. Column 1 is complete, and columns 0 and 2 are partially complete.

4 Your Task:

For each of the functions represented by a flowchart in this section, you will need to convert the flowchart to a **single Boolean expression** and return that expression as a result for that function. For functions without a flowchart, implement that as you would a normal function.

In all of the functions discussed in this section, it can be assumed that the board is represented as a 9-character long string, as discussed in Section 3.3, and that the input will be valid.

For the relevant functions, we also assume that player A is represented by `0`, player B is represented by `1`, and an empty cell is represented by a space.

Task	Functions	Marks
task1	getCell, makeMove, prettyPrint	25
task2	isValidMove, isInvalidMove, isPlayerTurn	12
task3	rowCheck, colCheck,	10
task4	isAWinner, isBWinner, boardFull gameOver	32
task5	checkAllRows, checkAllCols, upwardDiagonalCheck, downwardDiagonalCheck	17
testing		9

Table 1: Mark allocation

4.1 TicTacToeHelper

This file contains all the functions used to play a game of tic tac toe and is used by the provided `TicTacToe` file.

4.1.1 Functions

- string generateEmptyBoard()
  - This function should return a string representation of an empty board.
- char getCell(string board, int cellIndex)
  - This function should return the contents of the cell at the provided `cellIndex` using the passed-in board.
  - It can be assumed that both passed-in parameters are valid.

- string makeMove(std::string board, int position, char symbol)
  - This function should place the passed in `symbol` at the passed in `position` of the passed in `board` and return the modified `board`.
  - It can be assumed that the passed-in parameters will be valid.
  - For example, given:

```
board = "          "
position = 0
symbol = '0'
```

    The result is:

```
"0         "
```

- string prettyPrint(string board)
  - This function is used to represent the string encoded board as a proper tic tac toe board.
  - The output of this function should be formatted as follows:

```
<0>|<1>|<2>$-+-+-$<3>|<4>|<5>$-+-+-$<6>|<7>|<8>$
```

    where:
    - `<number>` represents the contents of the cell represented by the number. So `<0>` is the contents of cell 0.
    - `$` represents a newline.
    - The `|`, `+`, `-` are part of the formatting you should print out.
  - For example, consider the following input string:

```
"10 0 1 1 "
```

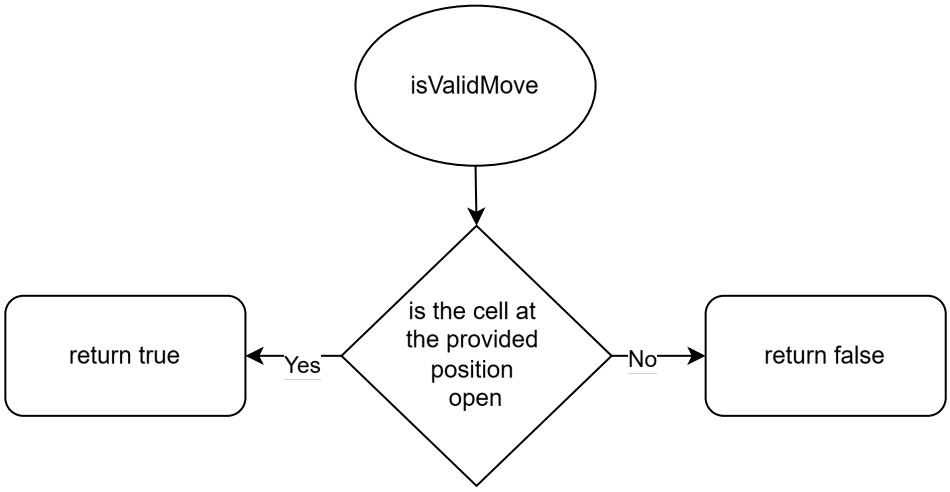
    The result of the function should be:

```
"1|0| \n-+-+-\n0| |1\n-+-+-\n |1| \n"
```

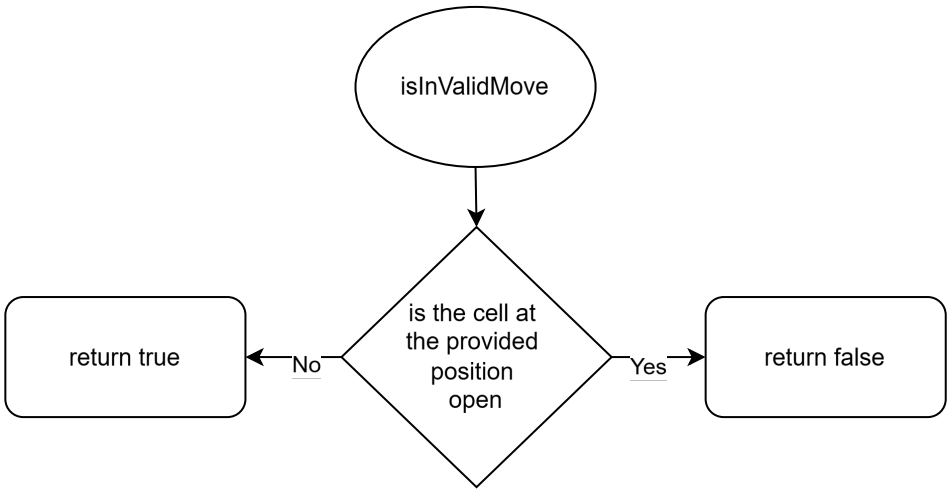
    Where `\n` indicates a newline. When this example is printed to console, the output should be:

```
1|0|
-+-+-
0| |1
-+-+-
 |1|
```

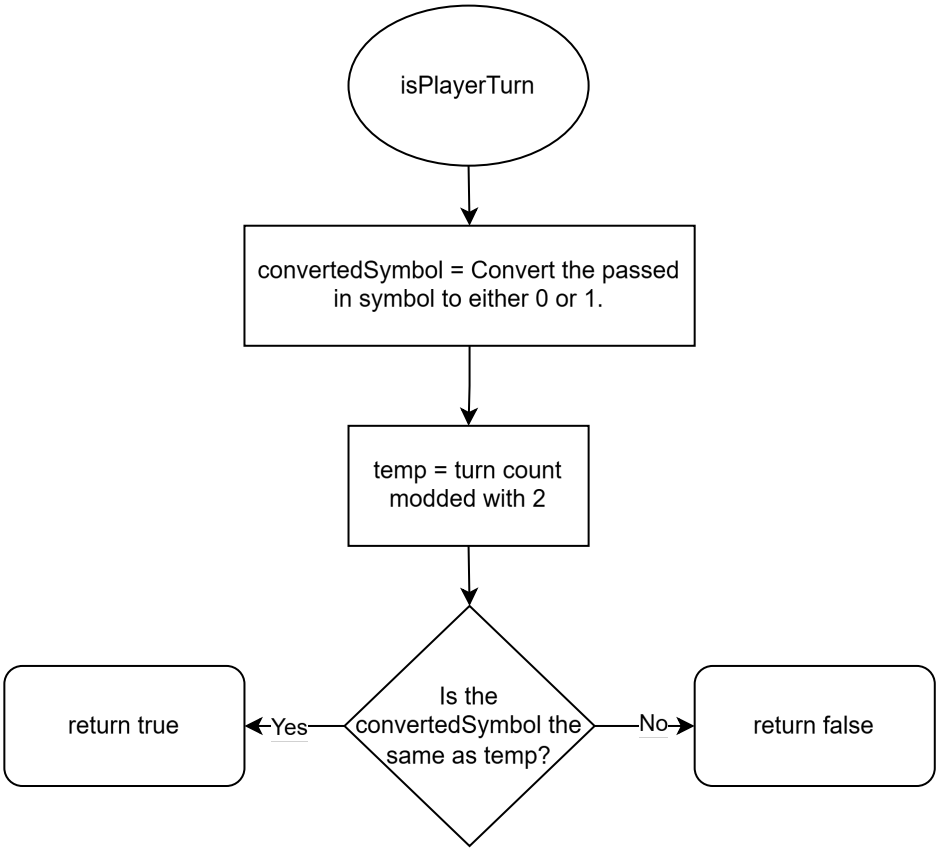
- bool isValidMove(string board, int cell)



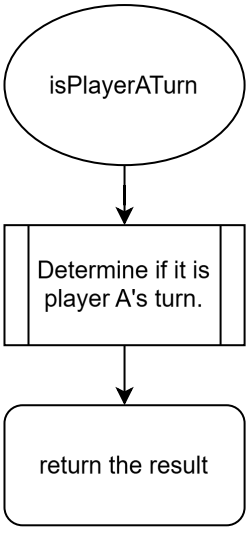
- bool isInValidMove(string board, int cell)



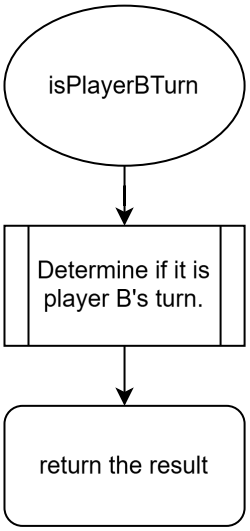
- bool isPlayerTurn(int turnCount, char symbol)



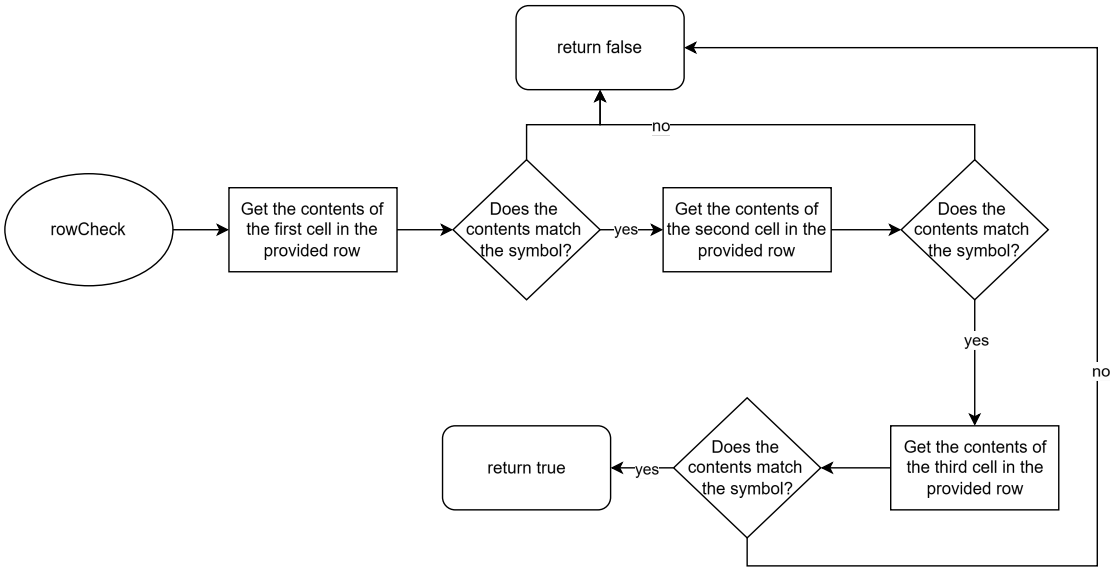
- bool isPlayerATurn(int turnCount)



- bool isPlayerBTurn(int turnCount)

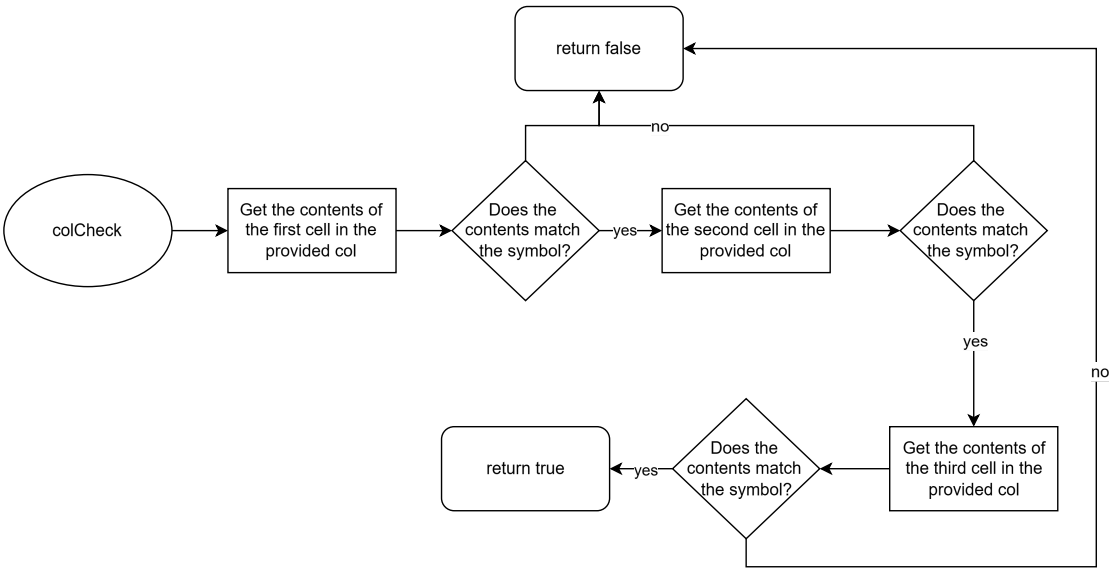


- bool rowCheck(std::string board, int row, char symbol)



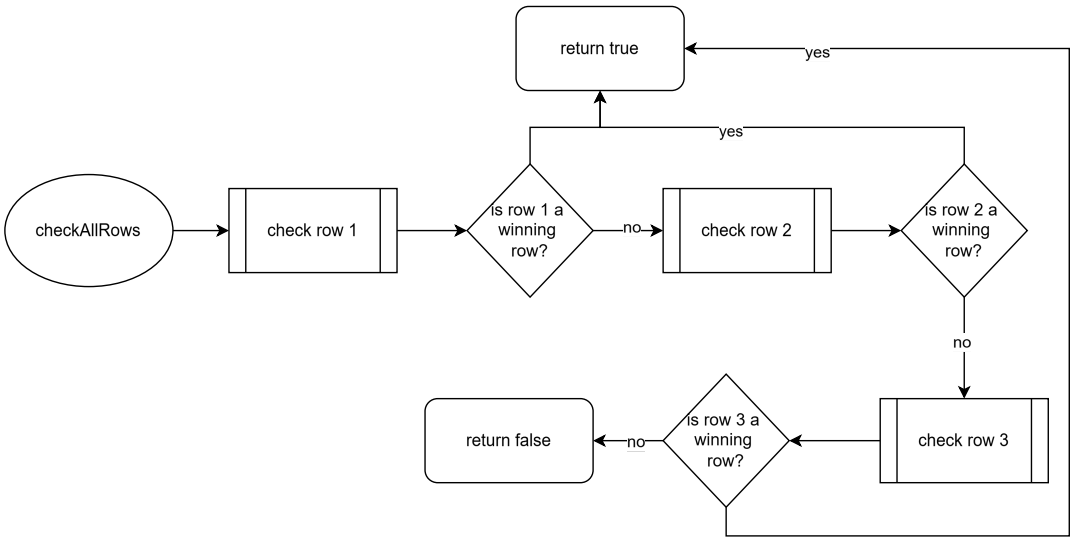
Hint: You will need to determine the calculations such that you can obtain the cells for a given row. Reminder, you do not need any repetition or selection structures.

- bool colCheck(std::string board, int col, char symbol)

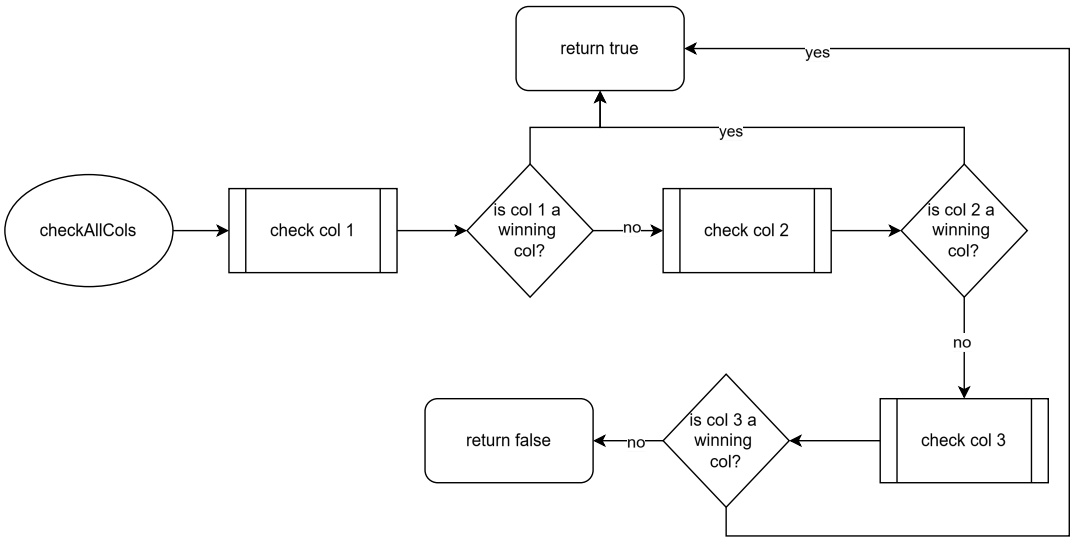


Hint: You will need to determine the calculations such that you can obtain the cells for a given row. Reminder, you do not need any repetition or selection structures.

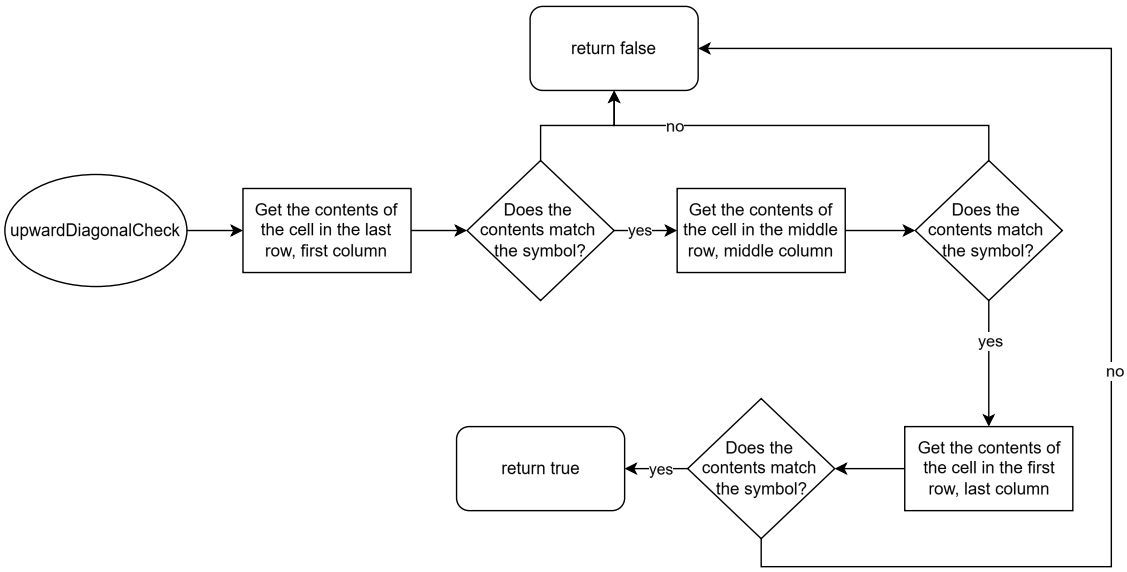
- bool checkAllRows(std::string board, char symbol)



- bool checkAllCols(std::string board, char symbol)

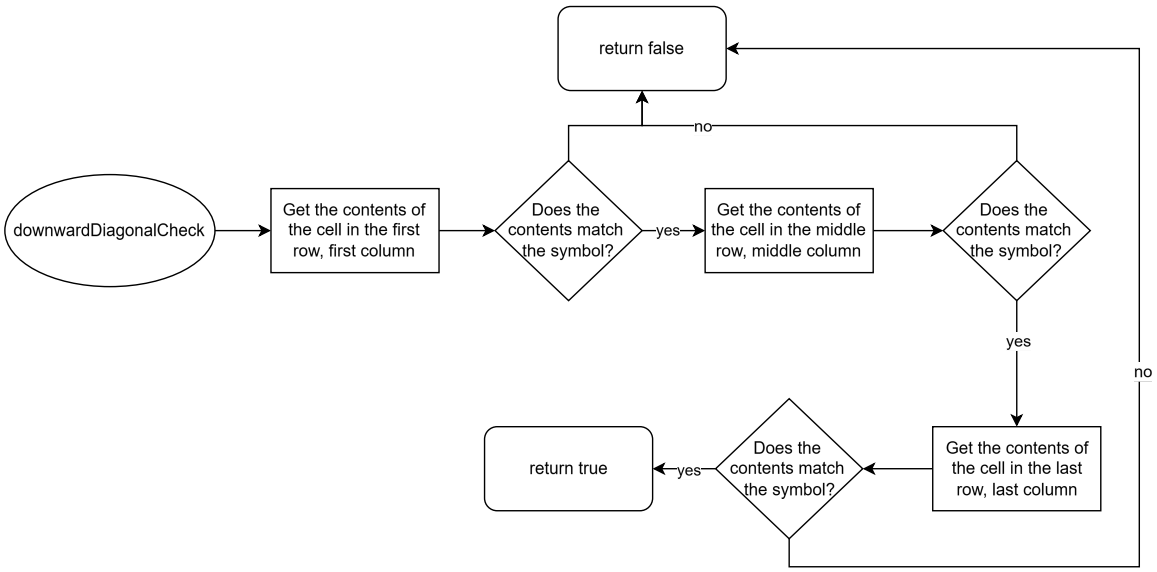


- bool upwardDiagonalCheck(std::string board, char symbol)



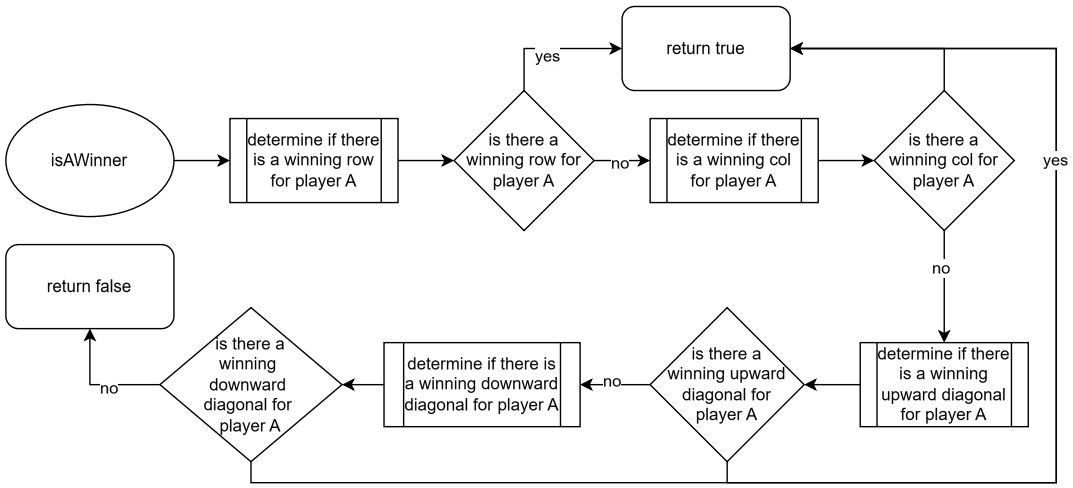
Hint: The positions of the cells used in this function can be determined while planning the algorithm.

- bool downwardDiagonalCheck(std::string board, char symbol)



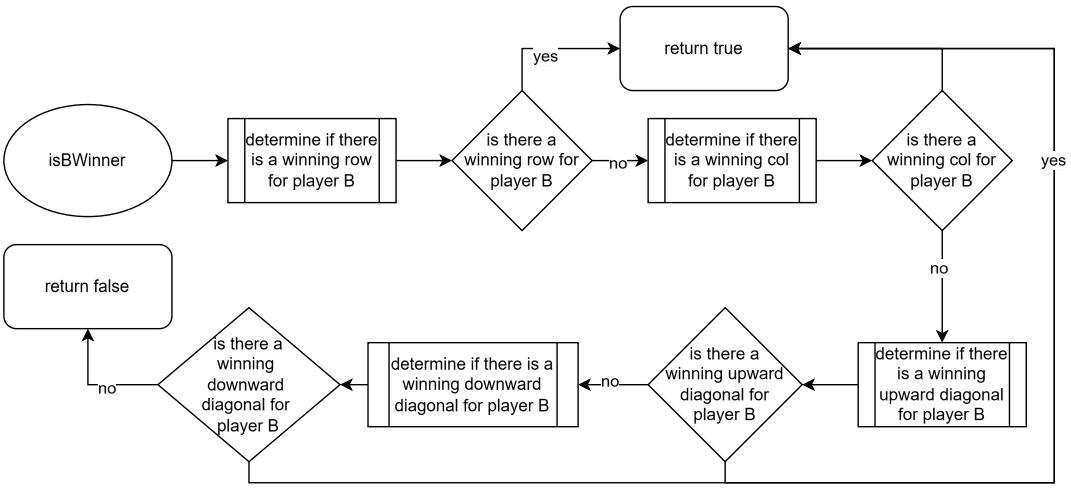
Hint: The positions of the cells used in this function can be determined while planning the algorithm.

- bool isAWinner(std::string board)



Hint: Remember player A is indicated by a `0`.

- bool isBWinner(std::string board)

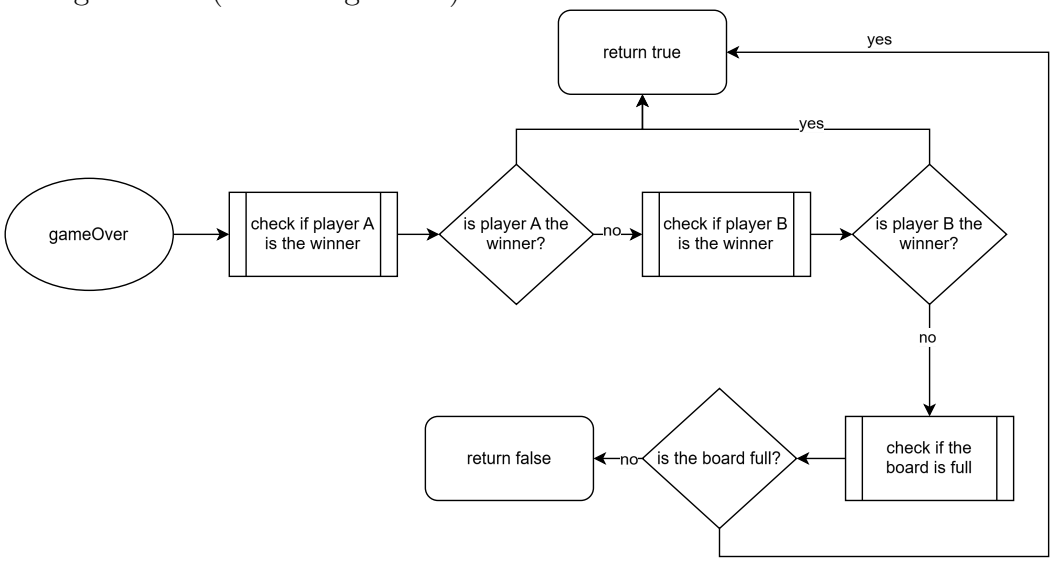


Hint: Remember player B is indicated by a 1.

- bool boardFull(std::string board)
  - The boardFull function's result can be expressed by the following expression:

$$\bigvee_{i=0}^{8} \text{cell}_i \; != \; `\ '$$

- bool gameOver(std::string board)



5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [2] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 2:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 2: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

[2] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **C++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following C++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- `TicTacToeHelper.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -std=C++98 -g *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 20 submissions and your final submission's mark will be your final mark. Upload your archive to the Practical 4 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**