Code Duello: The Ultimate Tic-Tac-Toe Showdown

Due: Sep 27, 2024 12:00 PM CST

Documentation Header Reminder

Before you start your assignment, you will need to add documentation similar to what we demonstrated in the first few lectures.

```
// Programmer: San Yeung
// Date: 9/4/1002
// File: fahr2celc.cpp
// Assignment: HW2
// Purpose: this file contains the main function of the program which
//    will input Fahrenheit temps from the user, then convert and output
//    the corresponding Celcius temperature.
```

Function Documentation Reminder

Having proper documentation for C++ functions is important. Make sure that each function includes the following:

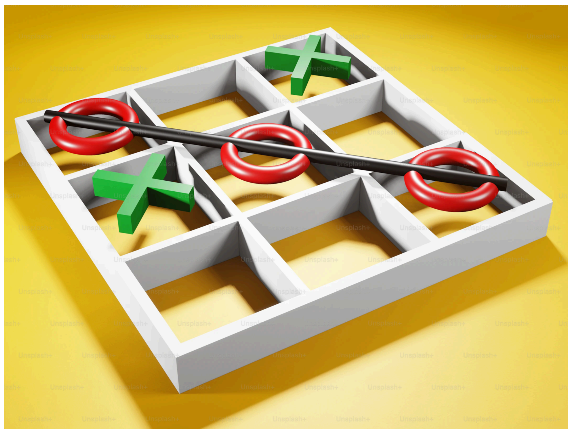
- General Description: A brief description of what the function does.
- Precondition: Describe any requirements that must be met before calling the function, e.g., function parameter conditions.
- Postcondition: Explain what the function accomplishes as the aftermath and report any significant changes it makes to persistent variables or data.

Objective: To reinforce key C++ programming concepts, including the use of switch-case statements for control flow, the implementation of basic functions, understanding and applying pass-by-value and pass-by-reference parameters to manage data effectively, and employing constant parameters in functions to ensure data integrity, through the designing and implementing a Tic-Tac-Toe game.

Background

In the digital realm of Tic Tac Terra, two ancient civilizations, the Circles and the Crosses, have coexisted in harmony for centuries. The game known as Tic-Tac-Toe was not just a pastime but a sacred ritual that brought balance to prosperity to Tic Tac Terra.

Today, you and your peers have been invited to partake in the Ultimate Showdown, a tournament designed to test your intellect, creativity, and coding prowess.



Program Specifications

In this assignment, your task is to develop a complete C++ program that presents a main menu to players, offering them the opportunity to play Tic-Tac-Toe of greater board size.

1. **Initial Prompt:** As the first step, your program should prompt the users to enter the first names of Player 1 and Player 2.
2. **Menu Options:** Upon starting, your program should display the following menu options (must be implemented using a switch-case) to the users:

```
Choose the size of the Tic-Tac-Toe board or Quit:
1. 3x3 Board
2. 4x4 Board
3. 5x5 Board
4. 6x6 Board
5. 7x7 Board
6. 8x8 Board
7. 9x9 Board
8. Quit the Program
```

Users will input a number to select their desired option. Your program should then proceed based on the user's selection. It is crucial that selecting an option outside the acceptable range will result in the program automatically terminating. Your implementation should reflect and ensure this feature.

Gameplay Implementation Details:

- The game board should be represented by a **single string variable**, with each cell's position indicated by two characters. Positions 1 through 9 will have a leading zero for consistency, e.g., "01", "02", ..., "09"). This setup allows players to easily choose their moves by selecting a corresponding cell number.
- In this special Missouri S&T edition, the symbols for the game will be 'SS' (for Player 1) and 'TT' (for Player 2), representing our beloved institution.
- **Turn Sequence and Board Display:** Players will take turns choosing positions on the board, marking them with their unique symbols.
    - At the beginning of each player's turn, display the current state of the board. The board should reflect the cell numbers to indicate unoccupied cells and the player symbols to indicate occupied cells.
    - Before a player's move is finalized, the program must verify that the chosen cell number is valid and not already occupied. If the chosen cell is not available, then the program should prompt for an alternative selection. This game should not allow players to **override** each other's moves and ensure fair game play.
    - Continue this process, alternating turns and displaying the updated board at the start of each turn until the game reaches a win condition or a draw.
- **Post-Game:** Once a win or draw is determined, display the final state of the board along with a message declaring the winner or announcing a draw.

Example: For example, if the players opt for a 4x4 board, then the initial assignment to the string should be the value "01020304050607080910111213141516". If Player 1 marks cell number 5, the string will be updated to "01020304SS0607080910111213141516". Following Player 1, if Player 2 decides to mark cell number 12, this action updates the board to "01020304SS060708091011TT13141516". To help visualize the board after these moves, one might imagine it as follows:

01	02	03	04
SS	06	07	08
09	10	11	TT
13	14	15	16

To ensure consistency, you are required to adhere to the specified, ordered numbering system for cell assignment within your program.

Example of Turn Sequence Output:

```
Current Board:
SS 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16

Player 2(TT), choose a cell: // take Player 2 cell input
```

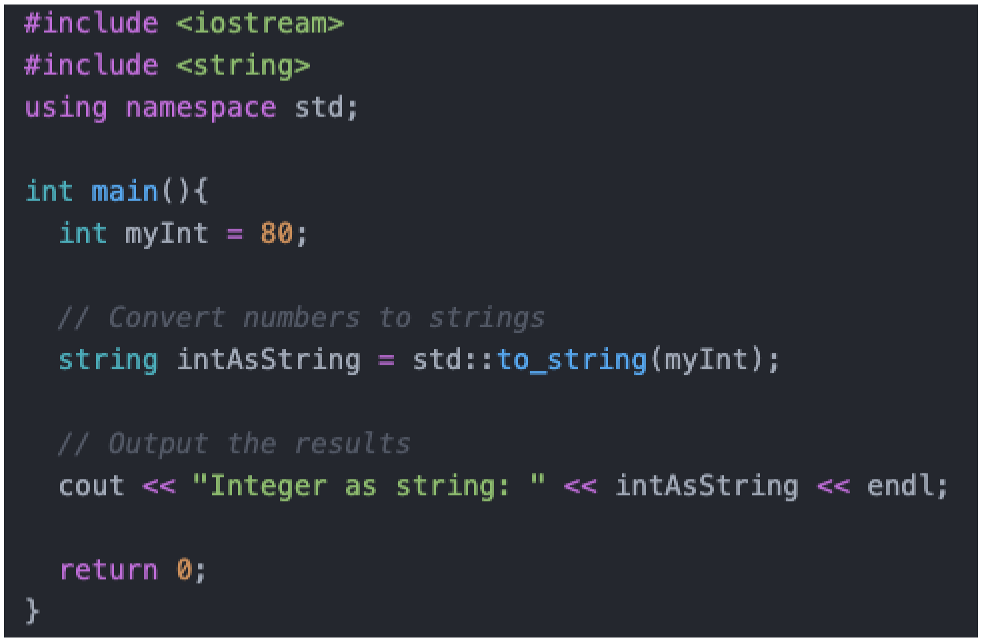
Winning & Tie Conditions:

- Winning this game is determined when a player successfully places their symbol across a **full** row, column, or along one of the two *major* diagonals of the board. This check should be performed *after* each player's move.
- A tie (or draw) occurs when all cells on the board are filled, and there is no winner. In this case, the game ends, and the program should announce a draw.
- Following the declaration of a win or a tie, the program should automatically return to the main menu and present the options anew, asking players to choose a new menu option.

Required Functions: The following specific, named functions are required. However, the choice of return type and parameters is left to your discretion. It will be your responsibility to decide on the most suitable function signatures based on your own implementation and design choices. So, be creative and effective in your solutions!

1. **`printWelcomeMessage()`:** Displays a welcome message at the beginning of the game, e.g., "Welcome to Dynamic Tic-Tac-Toe Missouri S&T Edition!").

2. **`displayMenu()`:** Shows the game's main menu, offering different board sizes or the option to quit.
3. **`initializeBoard()`:** Prepares the initial board string based on the selected size, filling it with numbered positions.
    a. **Suggested Signature:** `string initializeBoard(const int size);`
    b. **Tips:** Using [`std::to_string()`](std::to_string())can be incredibly useful here to help with initializing the game board. `std::to_string()`is a function in the C++ Standard Library `<string>` and can convert various numerical types (*int*, *long*, *float*, *double*, *etc.*) into a string.
        i. **Example:** Converting a number to a string using `std::to_string()`



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

int main(){
  int myInt = 80;

  // Convert numbers to strings
  string intAsString = std::to_string(myInt);

  // Output the results
  cout << "Integer as string: " << intAsString << endl;

  return 0;
}
```

The explicit `std::` prefix is recommended for standard library function calls to avoid potential name conflicts and to maintain good coding practice.

4. **`printBoard()`:** Prints the current state of the game board. Given the board's string representation and size, this function should format and display the board in a user-friendly manner.
    a. **Suggested Signature:** `void printBoard(const string& board, const int size);`

5. **`updateBoard()`:** Updates the board string based on the player's move. It takes the current board, the player-chosen cell position, and the player's symbol, and modifies the board string accordingly.
   a. **Suggested Signature:** `void updateBoard(string& board, const int size, const int chosenCell, const string& playerSymbol);`
6. **`checkWin()`:** Checks if the current board has a winning condition from any full row, column, or diagonal filled with the same player's symbol.
7. **`checkTie()`:** Checks if the game has ended in a tie. This happens when the board is full and there's no winner.
8. **`printGoodbyeMessage()`:** Displays a farewell message to the players as they exit the program. This can include encouragement to play again, or any other parting words you wish to convey.

Sample Output:

```
Welcome to Dynamic Tic-Tac-Toe Missouri S&T Edition!

Please enter the first name of Player 1: Googler
Please enter the first name of Player 2: Snickers

Choose the size of the Tic-Tac-Toe board or Quit:
1. 3x3 Board
2. 4x4 Board
3. 5x5 Board
4. 6x6 Board
5. 7x7 Board
6. 8x8 Board
7. 9x9 Board
8. Quit the Program
Enter your choice: 1

Current Board:
01 02 03
04 05 06
07 08 09
Player 1(SS), choose a cell: 01
```

```
Current Board:
SS 02 03
04 05 06
07 08 09
Player 2(TT), choose a cell: 02

Current Board:
SS TT 03
04 05 06
07 08 09
Player 1(SS), choose a cell: 04

Current Board:
SS TT 03
SS 05 06
07 08 09
Player 2(TT), choose a cell: 09

Current Board:
SS TT 03
SS 05 06
07 08 TT
Player 1(SS), choose a cell: 07

Current Board:
SS TT 03
SS 05 06
SS 08 TT

Congratulations Googler, you won!

Choose the size of the Tic-Tac-Toe board or Quit:
1. 3x3 Board
2. 4x4 Board
3. 5x5 Board
4. 6x6 Board
5. 7x7 Board
6. 8x8 Board
7. 9x9 Board
8. Quit the Program
Enter your choice: 8
```

```
Thank you for playing Dynamic Tic-Tac-Toe Missouri S&T Edition,
Giggles and Snickers!

Don't forget to Come back anytime for another round of fun and
games!
```

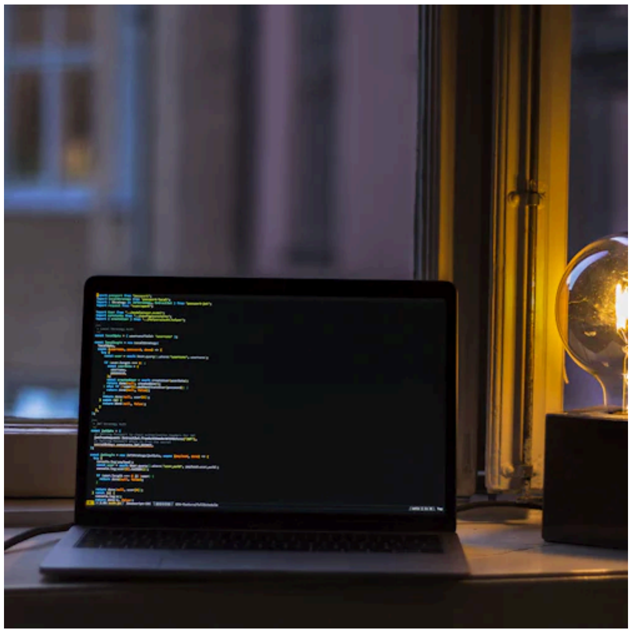
Important Notes

- If the parameters and return type of a function is not specified, then it is your responsibility to determine the most appropriate function signatures for them.
- You may create additional custom functions for this assignment as you see fit.
- You may use functions from other C++ libraries if 1) they are explicitly stated in the assignment or 2) the usage of the library functions has been thoroughly introduced in class. Otherwise, the usage of other functions is prohibited without the permission of your instructor.
- DO include input/range validations as long as they are appropriate. While input data type validation is not required for this assignment, it is good practice to validate the data values for range.
- Use modular programming! Break down the assignment into smaller functions that perform specific tasks. This will help make your code easier to read, test, and maintain.
- Thoroughly test each function before moving on to the implementation of the next function! This is referred to as the iterative/test-based development approach. It emphasizes on testing and refining each function before moving onto the next one. This helps to identify and fix errors early on, and ensure that the final program is reliable and robust.
- Test the program with different inputs. This will help you ensure that the program is functioning as expected and will also help you identify any bugs or errors.
- Read the error messages carefully if the program is not working as expected. Understanding the error messages will help you to identify and fix the problem.
- Don't forget to use an adequate amount of comments to explain your code and make it easy to understand.
- You must use proper indentation (two whitespaces) to make the code more readable and easy to understand.

- Don’t forget to have fun and be creative with your program design and implementation! Programming can be both challenging and interesting at the same time!
- Finally, don't hesitate to ask for help from the instructor or TA if you are having trouble with the assignment.

Grading Criteria:

1. Clarity and Understandability (20%)
    - Code should be easy to follow.
    - Variables are appropriately named and clearly defined.
    - Logical flow is evident.
2. Completeness (20%)
    - All components of the assignment are addressed.
    - Edge cases or unique scenarios are considered and handled appropriately.
3. Optimization and Efficiency (20%)
    - The solution takes advantage of efficient methods and avoids unnecessary steps.
4. Syntax and Structure (15%)
    - Code follows a consistent format and adheres to C++ standards.
    - Proper indentation is used to enhance code readability.
    - Use of appropriate C++ conventions and constructs.
    - Effective use of functions.
5. Problem-Solving and Logic (15%)
    - The solution effectively addresses the problem.
    - Logic used in the code is sound and free of major errors.
6. Documentation and Comments (5%)
    - Important steps are commented for clarity.
    - Complex sections of code have accompanying explanations.
7. Creativity and Originality (5%)
    - Students showcase original thinking.
    - Unique and efficient solutions to problems are encouraged.



Happy Coding!