AnsiTerminal Class

The AnsiTerminal class encapsulates functionalities for interacting with the terminal, allowing operations like printing text at specific locations, clearing the screen, and capturing key inputs.

Constructor: AnsiTerminal()

Purpose: This constructor is used to set up the terminal in non-canonical (raw) mode, allowing it to capture key presses immediately (without waiting for the user to press Enter).

Details: It saves the current terminal settings and disables canonical mode (input buffering) and echoing (displaying typed characters) for real-time key press handling.

Destructor: ~AnsiTerminal()

Purpose: The destructor restores the original terminal settings to ensure the terminal behaves normally after the program ends.

Details: It reverts the terminal back to the initial state (before any changes made by the constructor), restoring input buffering and character echoing.

Method: printAt(int row, int col, const std::string &text)

Purpose: Prints a given text at a specified row and column on the terminal screen.

Details: The method uses ANSI escape sequences to position the cursor at the specified row and column, then prints the provided text at that location.

Method: printInvertedAt(int row, int col, const std::string &text)

Purpose: Prints the text at the given row and column but with inverted background (reverse video mode).

Details: This method uses an ANSI escape sequence to enable the inverted color scheme, where text appears with the background and foreground colors swapped.

Method: clearScreen()

Purpose: Clears the terminal screen and resets the cursor position to the top-left corner.

Details: It sends an ANSI escape sequence that clears the screen and moves the cursor to the top-left position (home position).

Method: getKeystroke()

Purpose: Captures a single key press from the user.

Details: The method waits for the user to press a key and returns the corresponding character. If the user presses a control key (e.g., Ctrl+A), it returns a mapped readable character, and if the key is a comma (,), it returns the equal sign (=).

Method: getSpecialKey()

Purpose: Captures special keys like arrow keys or function key combinations (e.g., Alt+Key, Ctrl+Key).

Details: It checks for special key sequences, such as arrow keys (up, down, left, right) by reading the escape sequences. It returns corresponding single characters (e.g., 'U' for up arrow, 'D' for down arrow).

Summary of Behavior

Terminal Interaction: The AnsiTerminal class directly manages the terminal interface, allowing you to print text at specific locations, clear the screen, and interact with the user via keyboard inputs.

Keystroke Handling: Special handling for key combinations like Ctrl+A to Ctrl+Z and arrow keys ensures that even complex input scenarios can be captured accurately.

Screen Manipulation: The class is capable of printing text in different formats (regular or inverted) and cleaning up the terminal environment.

SpreadSheet

Constructor: Spreadsheet(int rows, int cols)

Spreadsheet::Spreadsheet(int rows, int cols)

: totalRows(rows), totalCols(cols), currentRow(1), currentCol(1), data(rows, std::vector<std::string>(cols)) {}

Purpose: This constructor initializes a new Spreadsheet object with the specified number of rows and columns.

totalRows and totalCols store the dimensions of the spreadsheet.

currentRow and currentCol track the position of the current active cell (starts at cell A1).

data is a 2D vector of strings that stores the content of each cell. It is initialized with empty strings.

Method: void display()

void Spreadsheet::display() {

...

}

Purpose: This method handles the display of the entire spreadsheet in the terminal.

Steps:

Display Current Cell: It shows the current active cell (A1, B5, etc.) and its content in the terminal.

Column Headers: Prints the column headers (e.g., A, B, C) in a specified format.

Row Numbers: Prints the row numbers (e.g., 1, 2, 3).

Cell Content: Displays the content of each cell in the grid, highlighting the current cell (using inverted colors for the active cell).

Formatting: It uses special terminal control codes to color the background of column and row headers, and handle the highlighting of the active cell.

Method: void moveCursor(char direction)

void Spreadsheet::moveCursor(char direction) {

...

}

Purpose: Moves the active cell cursor based on user input (up, down, left, right).

Steps:

Erase Old Cell: The current cell's content is erased from the terminal.

Check Boundaries: It checks if the new position is within the spreadsheet bounds (not exceeding the first row or last row, etc.).

Move Cursor: The cursor is moved in the specified direction (up, down, left, right).

Re-display: After moving the cursor, it calls display() to redraw the grid with the updated active cell.

Method: void editCell(const std::string& content)

void Spreadsheet::editCell(const std::string& content) {

...

}

Purpose: Allows editing the content of the currently active cell.

Steps:

Update Data: The content passed as an argument is appended to the current cell's existing data.

Re-display: After updating the cell, it calls display() to show the updated content.

Method: void evaluateFormula()

void Spreadsheet::evaluateFormula() {

...

}

Purpose: Evaluates the formula in the active cell if the cell contains a formula (indicated by a leading "=").

Steps:

Check Formula: It first checks if the content of the current cell starts with "=", which indicates a formula.

Parse Formula: The formula is parsed and evaluated. This method supports operations like addition, subtraction, multiplication, and division.

Handle Cell References: The formula can contain cell references (like A1, B2). These are resolved by accessing the data vector to fetch the values of the referenced cells.

Compute Result: The formula is evaluated step-by-step, and the result is stored in the current cell.

Handle Errors: If there is any issue (e.g., division by zero), the formula is evaluated to "Error".

Method: void formulas()

void Spreadsheet::formulas() {

...

}

Purpose: Handles special formulas like MAX, MIN, AVER, etc.

Steps:

Detect Formula Type: It first checks if the formula starts with a keyword like "MAX", "MIN", or "AVER".

Extract Range: The formula is parsed to extract the range of cells it applies to (e.g., A1-A10).

Evaluate Range: The method then calculates the desired result (e.g., finding the maximum value in the range, calculating the average, etc.).

Update Cell: After calculating the result, it updates the current cell with the computed value (e.g., the maximum or average).

Example of Formula Evaluation:

In the evaluateFormula() method, formulas that reference cells (e.g., =A1+B2) are parsed and evaluated. The method:

Extracts the cell references (like A1).

Converts the column letter to a column index (e.g., A to 0).

Retrieves the value from the data vector.

Evaluates the expression by applying the appropriate operators (addition, subtraction, etc.).

Handling Special Functions (MAX, MIN, etc.):

In the formulas() method, special functions are handled by:

Parsing the formula to get the range (e.g., A1-A10).

Iterating over the range to compute the maximum, minimum, or average, based on the function type.

For example, if the formula is MAX(A1-A10), the method:

Finds the starting and ending rows of the range (A1 to A10).

Computes the maximum value in that range and updates the current cell with the result.

Summary of Methods:

display(): Displays the entire spreadsheet and the content of each cell.

moveCursor(char direction): Moves the active cursor around the spreadsheet based on input.

editCell(const std::string& content): Edits the content of the active cell.

evaluateFormula(): Evaluates and updates the content of the active cell if it contains a formula.

formulas(): Handles and evaluates special formulas like MAX, MIN, and AVER in the active cell.







