Project - GW-Basic Programming Language Team - A

Program Interface Module Overview

Command Line Editor REPL System

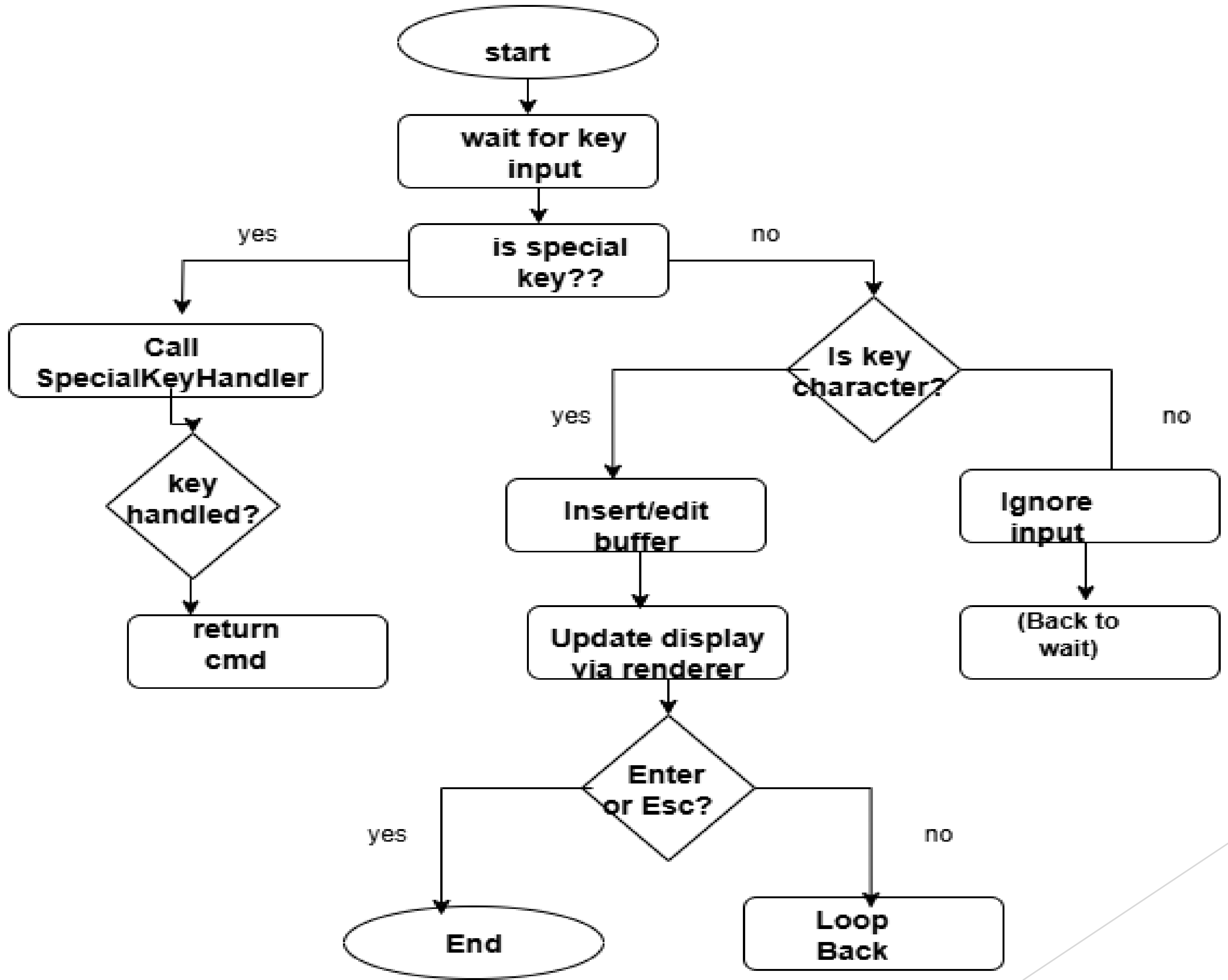
-Presented by

Vijaylaxmi(293952)

What is Program Interface?????

- Mimics classic GW-BASIC command line interface
- Handles input editing, cursor control, and special key commands
- Built using modular components:
- CommandLineEditor,
- ScreenRenderer,
- SpecialKeyHandler

Flow Diagram (Input Handling)



CommandLineEditor

Purpose:
Captures user input from the terminal, supports character insertion, deletion, and cursor navigation.

Key Members:

•std::string buffer: Current input line buffer.

•int cursorPosition: Current position of the cursor in the buffer.

•SpecialKeyHandler* specialKeyHandler: Optional handler for function keys (F1-F10).

Features:

•Live text editing

•Cursor movement with arrow keys

•Backspace handling

•Special key recognition

Screen Renderer

PURPOSE:
MANAGES THE DISPLAY OF THE CURRENT BUFFER ON THE TERMINAL.

INTERCEPTS SPECIAL KEYS (F1–F10, ARROW KEYS, ESC)

INJECTS COMMANDS LIKE RUN, LIST, LOAD, ETC. INTO BUFFER

RETURNS CONTROL IF A SPECIAL COMMAND IS RECOGNIZED

SpecialKeyHandler

Purpose: Maps function keys (F1–F10) to specific commands such as "RUN", "LIST", "NEW", etc.

Key Methods: handleSpecialKey(const check_SKey&, std::string&): If a known special key is pressed, sets a predefined command in the input buffer.

Handles redrawing the command buffer on screen

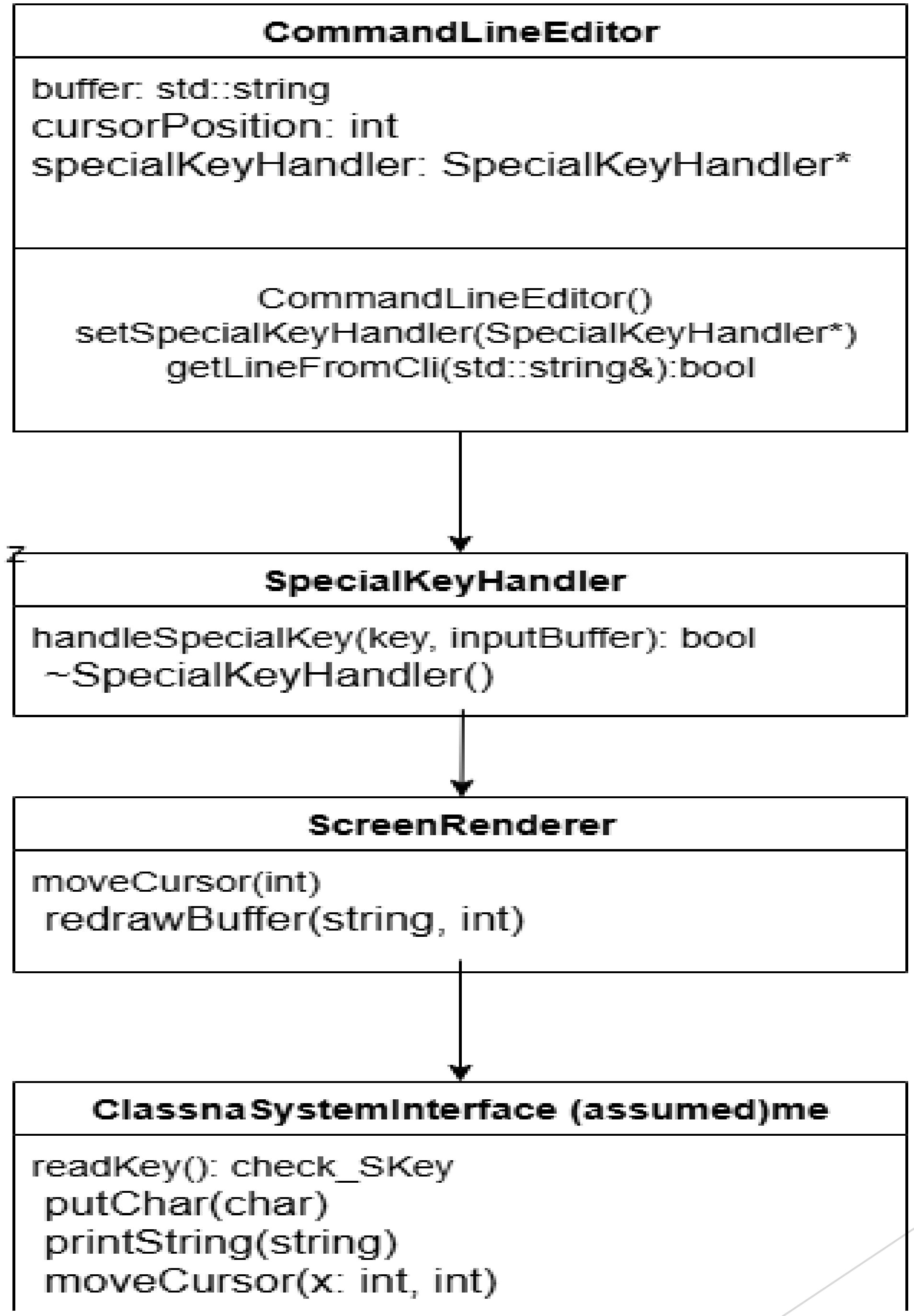
Moves cursor to correct position after edits

Static methods using SystemInterface for positioning

Function Key mapping

FunctionKey	injrected cmd	Console Effect
F1	RUN	Runs the current program
F2	LIST	Lists all program lines
F3	NEW	Clears program memory
F4	LOAD"	Prepares to load a program
F5	SAVE"	Prepares to save a program
F6	CONT	Continues from the last break
F7	LPT1	Sends output to printer (simulated)
F8	TRON	Enables trace mode
F9	TROFF	Disables trace mode
F10	KEY	Lists key mappings or shortcuts

Class diagram



Class Structure

- <u>CommandLineEditor</u>
- ├── getLineFromCli(std::string&): Reads user input interactively
- ├── SpecialKeyHandler(...): Attaches special key handle
- └──   Uses ScreenRenderer and SystemInterface

- <u>SpecialKeyHandler</u>
- └── handleSpecialKey(...): Handles function keys and injects command

- <u>ScreenRenderer</u>
- └── redrawBuffer(), moveCursor()

SystemInterface:

init(): Setup stub

readKey(): Reads a character or key

putChar(): Prints one char

getChar(): Reads one char using _getch()

printString(): Prints full string

clearScreen(): ANSI clear

moveCursor(x, y): Positions cursor

Internal Data Members

- *CommandLineEditor: - buffer:
- Current input - cursorPosition: Index in buffer - specialKeyHandler: Handles F1-F10
- check_SKey (Struct): - isSpecial: True if key is special - ch: Normal character - sKey: Enum for F1-F10, arrows

Future Enhancement

Tab Completion
•Implement a command/filename auto-completion on Tab.

Syntax Highlighting
•Color-code keywords, strings, or syntax while typing.

Multiline Editing
•Support for line wrapping and multiline input.

Mouse Support
•Handle mouse events for cursor movement and selection.

Command Validation
•Integrate a parser to provide live feedback on command correctness.

Modular Plugin System
•Allow users to inject their own special key behavior dynamically.

Console Interaction Example

```
shell
GW-BASIC64 v1.0
READY. > PRINT Hello World
 (user presses ← to fix typo)
> PRINT "Hello World" ↑
 (user presses → and ENTER) Hello World
 READY.
> 10 PRINT "HELLO“
 > 20 GOTO 10
 >LIST 10 PRINT "HELLO" 20 GOTO 10
 > RUN HELLO HELLO HELLO .
.. (user presses Ctrl+C or ESC to stop)
 READY.
 > SAVE "loop.bas" Saved successfully.
 > LOAD "loop.bas" Loaded: loop.bas
> NEW All program lines cleared. READY.
```

Conclusion

The **Program Interface module** in our GW-BASIC interpreter enables interactive user input with features like cursor movement, real-time editing, and backspace handling. It cleanly separates responsibilities across CommandLineEditor, ScreenRenderer, and SpecialKeyHandler. Function keys are mapped to BASIC commands for quick access. The design ensures modularity, ease of extension, and a classic BASIC feel. Overall, this module enhances

usability while maintaining maintainable and scalable code structure.

Thank you