GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

MARQUEE CONSOLE

PRESENTATION

CSOPESY REQUIREMENT

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

LIBRARIES USED

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <thread>
#include <chrono>
#include <atomic>
#include <mutex>
#include <iomanip>
#include <queue>
#include <windows.h>
#include <conio.h>
```

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

GLOBAL VARIABLES

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

```
// --- Shared State and Thread Control ---
std::atomic<bool> is_running{true};
std::atomic<bool> marquee_running{false}; // Marquee is stopped by default
std::string marquee_window = "";
std::atomic<int> marquee_speed{200}; // default 200 ms
std::string header = "Welcome to CSOPESY!\n\nGroup Developer:\nCastillo, Marvien Angel\nHerrera, Mikaela Gabrielle\nJimenez, Jaztin Jacob\nRegindin, Sean Adrien\n\nVersion Date: September 30, 2025\n";
std::string ascii_art = R"(
  ___ ___   ___  ____  ____   _____  __
 / __/ __| / _ \|  _ \|  __|/ __\ \/ /
| |  \__ \| | | | |_| |  _| \__ \\  /
| |__ __| | |_| |  __/| |___ __| | |
 \___|___/ \___/|_|   |_____|___/|_|

)";

std::string command_prompt = "Command > ";
```

```
/**
 * We will use three threads:
 * 1. Keyboard Handler Thread (reads user input)
 * 2. Marquee Logic Thread (updates marquee position)
 * 3. Display Thread (renders marquee and prompt)
 */

// The command interpreter and display thread share this variable.
std::string prompt_display_buffer = "";
std::mutex prompt_mutex;

// Shared state for the keyboard handler and command interpreter.
std::queue<std::string> command_queue;
std::mutex command_queue_mutex;

// The marquee logic thread and display thread share this variable.
std::string marquee_display_buffer = "This is a scrolling marquee text! ";
std::mutex marquee_to_display_mutex;

// Display parameters
int display_width = 40;
int marquee_pos = 0;

// Console mutex for ver4 functionality
std::mutex console_mutex;

// Line positions
const int MARQUEE_LINE = 17;
const int INPUT_LINE = 19;
const int OUTPUT_LINE = 21;
```

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

UTILITY FUNCTIONS

```cpp
// --- Utility Function ---
void gotoxy(int x, int y) {
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

// --- Functions to make it cleaner ---
void clear_line(int y, int width) {
    gotoxy(0, y);
    for (int i = 0; i < width; i++) {
        std::cout << " ";
    }
    std::cout << std::flush;
}

void clear_screen() {
    system("cls");
}

void hide_cursor() {
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = false;
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}

void show_cursor() {
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
    cursorInfo.bVisible = true;
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cursorInfo);
}
```

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

COMMAND RECOGNITION

CASTILLO HERRERA JIMENEZ REGINDIN

KEYBOARD HANDLER THREAD

The Keyboard Handler Thread reads the input of the users. When the program starts, the cursor will be put in the input line. This thread gets the text input of the user one character at a time, and only recognizes the end of the command when the enter key is pressed. If the user presses the backspace key, the last character will be deleted. It also ensures that the input is valid by checking if it is a printable ASCII character, it is within the max display length, and it is not empty. Once these are all validated, it will push the input into the command queue.

```
// --- Thread Functions ---
void keyboard_handler_thread_func() {
    const int max_display_length = 65;

    while (is_running) {
        {
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, INPUT_LINE);
            std::cout << "Command > " << std::flush;
        }

        std::string input = "";
        char ch;

        while (true) {
            if (_kbhit()) {
                ch = _getch();

                if (ch == '\r' || ch == '\n') {
                    // Enter key pressed
                    break;
                } else if (ch == '\b' || ch == 127) {
                    // Backspace
                    if (!input.empty()) {
                        input.pop_back();

                        std::lock_guard<std::mutex> console_lock(console_mutex);
                        gotoxy(10, INPUT_LINE);
                        std::cout << std::string(max_display_length, ' ') << std::flush;
                        gotoxy(10, INPUT_LINE);

                        std::string display_text = input;
                        if (display_text.length() > max_display_length) {
                            display_text = display_text.substr(0, max_display_length);
                        }
                        std::cout << display_text << std::flush;
                    }
```

(the rest of the function is in the next slide)

```
            } else if (ch >= 32 && ch <= 126) {
                // Printable character
                input += ch;

                std::lock_guard<std::mutex> console_lock(console_mutex);
                gotoxy(10, INPUT_LINE);

                std::string display_text = input;
                if (display_text.length() > max_display_length) {
                    display_text = "..." + display_text.substr(display_text.length() - max_display_length + 3);
                }

                std::cout << std::string(max_display_length, ' ') << std::flush;
                gotoxy(10, INPUT_LINE);
                std::cout << display_text << std::flush;
            }
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }

    if (!input.empty()) {
        {
            std::lock_guard<std::mutex> console_lock(console_mutex);
            clear_line(INPUT_LINE, 80);
        }
        std::unique_lock<std::mutex> lock(command_queue_mutex);
        command_queue.push(input);
    }
  }
}
```

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

CONSOLE UI IMPLEMENTATION

CASTILLO HERRERA JIMENEZ REGINDIN

DISPLAY THREAD

```
void display_thread_func() {
    const int refresh_rate_ms = 50;
    while (is_running) {
        {
            std::unique_lock<std::mutex> lock(prompt_mutex);
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, MARQUEE_LINE);
            // Clear the entire marquee line first
            clear_line(MARQUEE_LINE, 80);
            gotoxy(0, MARQUEE_LINE);
            std::cout << prompt_display_buffer << std::flush;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(refresh_rate_ms));
    }
}
```

The Display Thread is responsible for updating and rendering the console interface throughout the program's execution. It makes sure that the user sees the updated output. This thread operates independently from the other threads.

The Display Thread manages the console layout, including the input line, output display area, and marquee section. It ensures they are consistently updated with the program. The thread reads from shared buffers and queues, processes any new display data, and redraws the screen without interfering with input handling.

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

COMMAND INTERPRETER

CASTILLO HERRERA JIMENEZ REGINDIN

```
void marquee_logic_thread_func() {
    int i = 0;

    while (is_running) {
        if (marquee_running) {
            std::string text;
            {
                std::unique_lock<std::mutex> lock(marquee_to_display_mutex);
                text = marquee_display_buffer;
            }

            if (!text.empty()) {
                std::string padded = std::string(display_width, ' ') + text + std::string(display_width, ' ');

                // extract sliding window
                std::string window = padded.substr(i, display_width);

                // advance index
                i++;
                if (i > (int)padded.size() - display_width) {
                    i = 0; // restart scroll
                }

                // update display buffer
                {
                    std::unique_lock<std::mutex> lock(prompt_mutex);
                    prompt_display_buffer = window;
                }
            }
        } else {
            // Clear marquee when stopped
            std::unique_lock<std::mutex> lock(prompt_mutex);
            prompt_display_buffer = "[Marquee Stopped]";
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(marquee_speed));
    }
}
```

MARQUEE LOGIC THREAD

The Marquee Logic Thread maintains a marquee buffer and continuously shifts the text at fixed time intervals to simulate smooth scrolling. The updated text is written into the display buffer, allowing the Display Thread to render the updated marquee state. This thread runs independently from user input and command processing, ensuring that the marquee remains active and responsive even while other operations are being performed.

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

COMMAND PROCESS

The Command Process interprets and executes the user’s input after it has been captured by the Keyboard Handler Thread. It parses each command into a keyword and its arguments, then validates whether it is supported by the system. Once validated, the process performs the appropriate action, like set marquee text, set speed, and more. If the command is invalid, it generates an error message and passes it to the display buffer.

```cpp
void command_process(const std::string& command_line) {
    std::istringstream iss(command_line);
    std::string cmd;
    iss >> cmd;

    // Clear ALL output areas more thoroughly
    {
        std::lock_guard<std::mutex> console_lock(console_mutex);
        for (int i = OUTPUT_LINE; i <= OUTPUT_LINE + 10; i++) {
            clear_line(i, 100);  // Clear wider area
        }
    }

    if(cmd == "help"){
        std::lock_guard<std::mutex> console_lock(console_mutex);
        gotoxy(0, OUTPUT_LINE);
        std::cout << "Available commands:" << std::flush;
        gotoxy(0, OUTPUT_LINE + 1);
        std::cout << "  help          - show this help message" << std::flush;
        gotoxy(0, OUTPUT_LINE + 2);
        std::cout << "  start_marquee - start marquee animation" << std::flush;
        gotoxy(0, OUTPUT_LINE + 3);
        std::cout << "  stop_marquee  - stop marquee animation" << std::flush;
        gotoxy(0, OUTPUT_LINE + 4);
        std::cout << "  set_text      - set marquee text" << std::flush;
        gotoxy(0, OUTPUT_LINE + 5);
        std::cout << "  set_speed     - set marquee animation speed (ms)" << std::flush;
        gotoxy(0, OUTPUT_LINE + 6);
        std::cout << "  exit          - exit emulator" << std::flush;
    } else if(cmd == "start_marquee"){
        marquee_running = true;
        std::lock_guard<std::mutex> console_lock(console_mutex);
        gotoxy(0, OUTPUT_LINE);
        std::cout << "Marquee started!" << std::flush;
    } else if(cmd == "stop_marquee"){
        marquee_running = false;
        std::lock_guard<std::mutex> console_lock(console_mutex);
        gotoxy(0, OUTPUT_LINE);
        std::cout << "Marquee stopped!" << std::flush;
    } else if (cmd == "set_text") {
        std::string new_text;
        std::getline(iss, new_text);
        if (!new_text.empty() && new_text[0] == ' ') new_text.erase(0, 1);
        if (!new_text.empty()) {
            std::unique_lock<std::mutex> lock(marquee_to_display_mutex);
            marquee_display_buffer = new_text + " ";
            lock.unlock();

            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, OUTPUT_LINE);
            std::cout << "Text set to: " << new_text << std::flush;
        } else {
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, OUTPUT_LINE);
            std::cout << "Usage: set_text <your text here>" << std::flush;
        }
    } else if (cmd == "set_speed") {
        int speed;
        if (iss >> speed && speed > 0) {
            marquee_speed = speed;
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, OUTPUT_LINE);
            std::cout << "Speed set to: " << speed << " ms" << std::flush;
        } else {
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, OUTPUT_LINE);
            std::cout << "Invalid speed." << std::flush;
        }
    } else if (cmd == "exit") {
        is_running = false;
        {
            std::lock_guard<std::mutex> console_lock(console_mutex);
            gotoxy(0, OUTPUT_LINE);
            std::cout << "Exiting program. Goodbye!" << std::flush;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
        exit(0);
    } else if (!cmd.empty()) {
        std::lock_guard<std::mutex> console_lock(console_mutex);
        gotoxy(0, OUTPUT_LINE);
        std::cout << "Unknown command. Type 'help' for a list of commands." << std::flush;
    }
}
```

COMMAND PROCESS

CASTILLO HERRERA JIMENEZ REGINDIN

COMMAND INTERPRETER IN MAIN

The Command Interpreter in Main initializes all required threads, including the Keyboard Handler, Display Thread, and Marquee Logic Thread, and sets up the shared data structures like the command queue and display buffer. Within its main loop, it continuously waits for new commands from the queue, processes them using the Command Process, and updates the display accordingly. It also ensures synchronization between threads to prevent conflicts, and monitors for termination commands such as exit to gracefully shut down the system.

CASTILLO HERRERA JIMENEZ REGINDIN

MAIN

```cpp
int main() {
    // Setup console
    hide_cursor();

    //COMMAND INTERPRETER
    clear_screen();
    std::cout << header << ascii_art << std::flush;

    std::thread marquee_logic_thread(marquee_logic_thread_func);
    std::thread display_thread(display_thread_func);
    std::thread keyboard_handler_thread(keyboard_handler_thread_func);

    while(is_running) {
        std::string command_line;
        {
            std::unique_lock<std::mutex> lock(command_queue_mutex);
            if (!command_queue.empty()) {
                command_line = command_queue.front();
                command_queue.pop();
            }
        }

        if (!command_line.empty()) {
            command_process(command_line);
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }

    // Join threads
    if (marquee_logic_thread.joinable()) marquee_logic_thread.join();
    if (display_thread.joinable()) display_thread.join();
    if (keyboard_handler_thread.joinable()) keyboard_handler_thread.join();

    // Restore cursor
    show_cursor();
    gotoxy(0, 35);
    std::cout << "\n\nGoodbye!\n";
    return 0;
}
```

GROUP 4

CASTILLO HERRERA JIMENEZ REGINDIN

THANK YOU