Name : Tushar Shivaji Kumbhar Roll no:325

❖ Title

Countdown Timer using C++.

❖ Objective

The objective of this project is to develop a C++ program that functions as both a Countdown Timer and a Stopwatch. Users can either set a specific time to count down from or start a stopwatch that keeps track of elapsed time. The aim is to enhance time-tracking ability through a simple command-line tool using object-oriented programming in C++.

❖ Methodology / Approach

The project is built using object-oriented programming principles in C++, with a focus on modularity, reusability, and user interaction via the command-line interface. The methodology followed in the development of this project includes the following steps:

1. Class Design Using Inheritance

- A base abstract class named Timer is defined with a pure virtual function start().
- This class also includes two utility functions:
    - convertSecondsToHumanReadableTime(double seconds): Converts total seconds into hours, minutes, and seconds format.
    - calculateTotalSeconds(double hours, double minutes, double seconds): Converts a time input into total seconds.
- Two derived classes inherit from Timer:
    - CountdownTimer: Implements countdown logic.
    - StopwatchTimer: Implements stopwatch functionality.

2. Countdown Timer Functionality

- When the user selects the countdown timer:
    - The program prompts for hours, minutes, and seconds.
    - It calculates total seconds using the helper method from the base class.
    - A while loop is used to decrement the timer by 1 second intervals.
    - After each second, the current time remaining is displayed in human-readable format.
    - When the countdown ends, a message "Time's up!" is shown.
    - The user is prompted to either reset the timer or quit the program.

3. Stopwatch Functionality

- If the user selects the stopwatch:

- The current time is captured using time(NULL) to record the starting time.
- An infinite loop continuously calculates and displays elapsed time.
- The user is informed to stop the stopwatch using Ctrl+C.
- Signal handling (SIGINT) is implemented using the signal() function.
- When the user presses Ctrl+C, the signal handler sets a flag that breaks the loop, allowing the stopwatch to stop gracefully and display a final message.

4. Real-Time Timing

- The usleep(1000000) function is used to pause the loop for 1 second (1,000,000 microseconds).
- This ensures accurate 1-second updates for both countdown and stopwatch displays.
- Real-time display is achieved by overwriting the same console line using carriage return (\r) and flush.

5. Signal Handling for Graceful Exit

- A volatile sig_atomic_t flag is defined to detect SIGINT.
- When Ctrl+C is pressed, a signal handler sets this flag.
- The main stopwatch loop checks this flag on every iteration to exit cleanly.
- This prevents abrupt termination and improves user experience.

6. Compilation and Execution Using Makefile

- A simple Makefile is created to automate the compilation process.
- The make command compiles main.cpp using g++ and creates an executable file named timer.
- The make run command runs the program, and make clean removes compiled files.
- This simplifies testing and avoids manual compilation steps.

7. Testing and Validation

- The project was tested for various time inputs including:
  - Short countdowns (e.g., 5 seconds)
  - Full hour-minute-second combinations (e.g., 1 hour 15 minutes 30 seconds)
  - Stopwatch accuracy and responsiveness to Ctrl+C
- It passed tests for both countdown and stopwatch functionalities, including edge cases like 0-second countdowns and invalid time entries.

Name : Tushar Shivaji Kumbhar Roll no:325

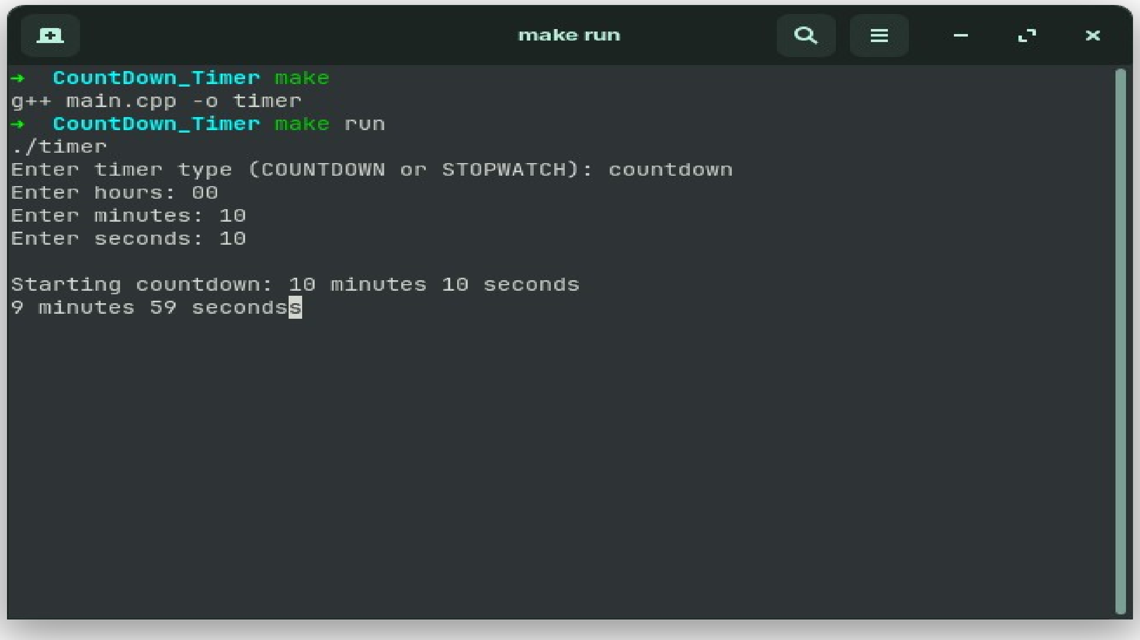
❖ Key Features

- Interactive command-line interface
- Supports both countdown and stopwatch modes
- Object-oriented design with inheritance
- Real-time time tracking using usleep()
- Graceful exit using signal handling for stopwatch
- Easy to compile with a Makefile

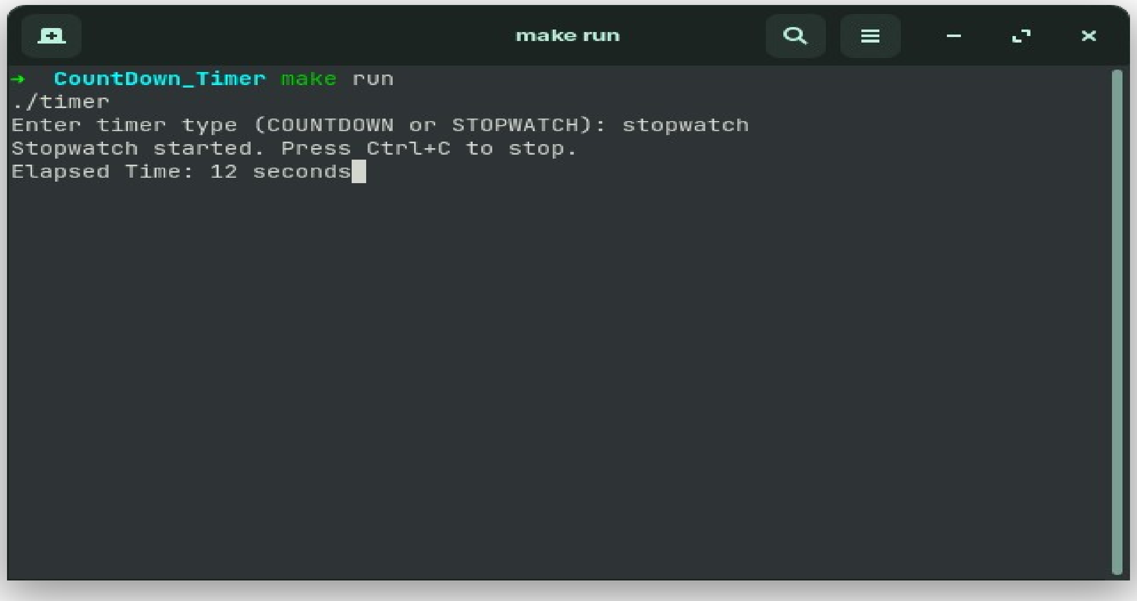
❖ Code

```
#include <iostream>
#include <string>
#include <unistd.h>
#include <ctime>
#include <iomanip>

using namespace std;

class Timer {
public:
    virtual void start() = 0;
    virtual ~Timer() {}

protected:
    string convertSecondsToHumanReadableTime(double seconds) {
        if (seconds < 0) {
            return "Invalid time!";
        }

        int hrs = seconds / 3600;
        seconds -= hrs * 3600;
        int mins = seconds / 60;
        seconds -= mins * 60;
        int secs = static_cast<int>(seconds);

        stringstream ss;
        if (hrs > 0) ss << hrs << " hours ";
        if (mins > 0 || hrs > 0) ss << mins << " minutes ";
        ss << secs << " seconds";
        return ss.str();
    }

    double calculateTotalSeconds(double hours, double minutes, double seconds) {
        if (hours < 0 || minutes < 0 || seconds < 0) {
            cout << "Please ensure all inputs are non-negative.\n";
            return 0;
        }
        return (hours * 3600) + (minutes * 60) + seconds;
    }
};

class CountdownTimer : public Timer {
public:
    CountdownTimer(double h, double m, double s) : hours(h), minutes(m), seconds(s) {}

    void start() override {
        double totalSeconds = calculateTotalSeconds(hours, minutes, seconds);
        cout << "\nStarting countdown: " << convertSecondsToHumanReadableTime(totalSeconds)
<< "\n";

        while (totalSeconds > 0) {
            cout << "\r" << convertSecondsToHumanReadableTime(totalSeconds) << flush;
            usleep(1000000);  // 1 second
            totalSeconds--;
        }

        cout << "\nTime's up!\n";
        handleReset();
    }

private:
    double hours, minutes, seconds;

    void handleReset() {
        char input;
        cout << "Would you like to RESET (R) or QUIT (q)? ";
        cin >> input;

        if (input == 'R' || input == 'r') {
            start();
        }
    }
};

class StopwatchTimer : public Timer {
public:
    void start() override {
        time_t startTime = time(NULL);
        cout << "Stopwatch started. Press Ctrl+C to stop.\n";

        while (true) {
            time_t elapsed = time(NULL) - startTime;
            cout << "\rElapsed Time: " << convertSecondsToHumanReadableTime(elapsed) << flush;
            usleep(1000000);
        }
    }
};

int main() {
    string timerType;
    cout << "Enter timer type (COUNTDOWN or STOPWATCH): ";
    cin >> timerType;

    Timer* timer = nullptr;

    if (timerType == "COUNTDOWN" || timerType == "countdown") {
        double h, m, s;
        cout << "Enter hours: ";
        cin >> h;
        cout << "Enter minutes: ";
        cin >> m;
        cout << "Enter seconds: ";
        cin >> s;

        timer = new CountdownTimer(h, m, s);
    } else if (timerType == "STOPWATCH" || timerType == "stopwatch") {
        timer = new StopwatchTimer();
    } else {
        cout << "Invalid timer type selected.\n";
        return 1;
    }

    timer->start();
    delete timer;
    return 0;
}
```

❖ Result / Output



```
→  CountDown_Timer make
g++ main.cpp -o timer
→  CountDown_Timer make run
./timer
Enter timer type (COUNTDOWN or STOPWATCH): countdown
Enter hours: 00
Enter minutes: 10
Enter seconds: 10

Starting countdown: 10 minutes 10 seconds
9 minutes 59 secondss
```



```
→  CountDown_Timer make run
./timer
Enter timer type (COUNTDOWN or STOPWATCH): stopwatch
Stopwatch started. Press Ctrl+C to stop.
Elapsed Time: 12 seconds
```

❖ Conclusion

This project demonstrates the use of C++ to build a real-time utility tool that leverages classes, inheritance, and system-level functions like sleep and signal handling. It serves as a strong example of object-oriented programming and can be extended further with GUI, sound alerts, or mobile support for broader usability.