***ACTIVITY FOR GROUP 6:***

***TOPIC: Reminder system***

**Group members:**

**1) Vedantika Ranaware – B24CE1013**

**2) Rugved Deshwant – B24CE1014**

**3) Swarada Deshpande - B24CE1015**

**4) Janavhi Dhanapune - B24CE1016**

**RESEARCH**

**Problem Statement**: The goal is to create a reminder system that alerts the user at regular intervals. This reminder should be sent a specified number of times, with a 60-second delay between each reminder.

 **Recursion**: The task will be implemented using recursion, where the function sendReminder calls itself to handle sending multiple reminders.

 **Sleep Function**: To simulate a delay between reminders, the sleep() function is used to pause the execution for 60 seconds.

 The program needs to handle reminders efficiently and stop at the appropriate time.

 We need to consider the performance of recursion and the delay caused by sleep(60) in between reminders.

**ANALYSE**

**Function Design**: The sendReminder(int count) function is recursive. It accepts the number of reminders (count) as its argument.

* **Base Case**: When count reaches zero, the function terminates and prints a final message "All reminders sent!".
* **Recursive Case**: If count is not zero, it prints a reminder message and waits for 60 seconds using the sleep(60) function before calling itself with a decremented count (count - 1).

**Flow**:

1. The main function calls sendReminder() with the initial reminder count.
2. The recursive function executes, printing reminders and waiting for 60 seconds before invoking itself with a reduced count.
3. When count reaches 0, the recursion stops, and the final message is printed.

**Edge Cases**:

* **Zero reminders**: If totalReminders is set to 0, no reminder is printed, and the message "All reminders sent!" will be immediately displayed.
* **Negative count**: If a negative count is passed, the recursion would not behave as intended, as it would keep calling itself indefinitely with a negative number. This can be prevented by ensuring the count parameter is positive.

The header files *<stdio.h>, <unistd.h>, <stdbool.h>, <signal*.h>, and <stdlib.h> are part of the C standard library and are used for different purposes:

* **<stdio.h>**: The standard input/output header file that contains functions for input and output operations, such as printf(), scanf(), fopen(), fclose(), and fseek().
* **<stdlib.h>**: A header file that contains a variety of utility functions, such as memory allocation functions (malloc(), calloc(), and free()), process control functions (abort() and exit()), and conversions functions (atoi() and atol()).
* **<unistd.h>**: A nonstandard C header that supports a variety of options, including real-time signals, semaphores, shared memory objects, and more.
* **<stdbool.h>**: A header file that defines a Boolean data type.
* **<signal.h>**: A header file that defines signal-handling functions.

Header files are used to reduce the complexity and number of lines of code in a program. The #include directive inserts a copy of the header file into a file before compilation.

**IDEATE**

In this stage, we think about possible improvements and alternatives:

* **Optimization**: While recursion is neat, it uses stack space. For larger counts (e.g., 100+ reminders), it may lead to a stack overflow. To avoid this, an iterative approach using a loop could be more appropriate.
* **Flexibility**: Instead of hardcoding the 60-second interval, we could allow the user to specify the interval for each reminder.
* **User Experience**: The user may not want to wait 60 seconds between reminders in a real-world application. For testing purposes, we could reduce the sleep time or make it configurable.
* **Possible Enhancements**:
  + Adding a user interface for interaction (e.g., allowing users to input the number of reminders and the interval).
  + Implementing a logging mechanism to track the reminder times.
  + Setting the program to run in the background and notify the user via sound or pop-up.

**BUILD**

**PROGRAM:**

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <stdlib.h>

int totalReminders = 0;

void stopReminders(int sig) {

printf("\nTotal reminders sent: %d\n", totalReminders);

exit(0);

}

void sendReminder() {

while (1) {

printf("Reminder: Time to check your task!\n");

totalReminders++;

sleep(10);

}

}

int main() {

signal(SIGINT, stopReminders);

printf("Welcome to the reminder system!\n");

printf("Press Ctrl+C to stop the reminders at any time.\n");

sendReminder();

return 0;

}

**TESTING**

Testing ensures that the solution works as expected. Here are the test case:

Welcome to the reminder system!

Press Ctrl+C to stop the reminders at any time.

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reminder: Time to check your task!

Reached 10 reminders, stopping now.

Program has finished sending reminder.

**IMPLEMENT**

 **Key Functions**:

* sendReminder(int count): Sends a reminder and waits for the specified interval.
* sleep(60): Introduces a 60-second delay between reminders.

 **Expected Behavior**: The program will print a reminder every 60 seconds for the number of times specified in totalReminders. The base case ensures that the recursion terminates once all reminders are sent.

### Conclusion

This reminder system, implemented with recursion, successfully fulfills the task of sending periodic reminders with delays. While recursion provides a simple approach, optimizations for larger input sizes and improvements to user experience could be considered for practical usage.

Reference links:

<https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm>

<https://www.geeksforgeeks.org/signals-c-language/>

<https://www.geeksforgeeks.org/sleep-function-in-c/>