Bootcamp on "Critical Thinking & Problem Solving

TEAM NAME:SEGMENTATION FAULT

S.NO	NAME	USN
1	Nandini Hegde	4MT22CI017
2	Marushka Rachelle D'silva	4MT22CI029
3	Nischal K	4MT22CI036
4	Riyaz K	4MT22CI041
5	Sujal Revankar	4MT22CI055
6	Vikas M	4MT22CI060
7	Yashita Vasu	4MT22CI062

CONTENTS

INTRODUCTION

The Gregorian calendar is the calendar used in most parts of the world.The principal change was to space leap years differently so as to make the average calendar year 365.2425 days long, more closely approximating the 365.2422-day ‘tropical’ or ‘solar’ that is determined by the Earth's revolution around the Sun. It serves as the benchmark for time measurement and facilitates the organization of daily activities, events, and holidays. The Gregorian Calendar serves as the international standard for time measurement, facilitating seamless communication and scheduling across different countries and time zones. Its universal adoption ensures efficient coordination in various domains, including business, travel, and diplomacy

HOW IT IS CALCULATED?

The rule for leap years is:

Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the year 2000 is.

We take the reference year as 1900. We ask the user to enter a year between 1900 and 2099 and store it inside the address of the year variable. Now we find the difference (1900 – year) and store it inside variable diff.

Using a while loop, we loop through until the reference year(1900) is less than the user entered year. Inside the while loop we check for all the leap years present from 1900 to the user entered year.

Total_days = (diff – leap) x 365 + leap x 366;

Now every year has 12 months, but each month has a different number of days. But every week has exactly 7 days. So we divide Total_days by 7 and store the reminder inside variable day.

PROJECT DESCRIPTION

Gregorian calendar provides a calendar view of a particular month of a specified year by taking them as inputs from the user into the program. The Personal Calendar Application is a user-friendly software tool designed to help individuals organize their schedules, plan events, and manage their time efficiently using the Gregorian calendar system. Key features include interactive calendar interface, event management, reminder notifications, categories and labels, search and filter capabilities, synchronization across devices, data backup and restore, and customization options.
Target Audience: Professionals, students, parents, freelancers, and individuals seeking effective schedule management.

USE CASE

Gregorian calendar serves as a versatile tool for users across various domains, facilitating organization, coordination, and planning in both personal and professional contexts such as:

1. Personal Planning: Users can use the Gregorian calendar to plan their personal schedules, including appointments, birthdays, anniversaries, and other important events. By visualizing their time on a calendar, users can efficiently manage their daily, weekly, and monthly activities.
2. Work Scheduling: In a professional setting, employees can use the Gregorian calendar to schedule meetings, deadlines, project milestones, and other work-related events. It helps them coordinate with colleagues, prioritize tasks, and allocate time effectively.
3. School and Academic Planning: Students and educators rely on the Gregorian calendar to organize their academic schedules, including class timings, exam dates, holidays, and study sessions. It enables them to stay on track with their coursework and manage their academic responsibilities efficiently.
4. Travel Planning: When planning vacations or business trips, users often refer to the Gregorian calendar to check dates, duration, and availability. They can schedule flights, hotel reservations, and activities accordingly, taking into account factors such as holidays, seasons, and weather conditions.
5. Financial Management: Users utilize the Gregorian calendar to track billing cycles, payment due dates, budgeting, and financial planning. By marking important financial events on the calendar, such as bill payments, income deposits, and investment deadlines, users can manage their finances more effectively.
6. Health and Fitness Tracking: Individuals interested in maintaining a healthy lifestyle can use the Gregorian calendar to schedule workouts, track progress, set fitness goals, and

monitor health-related activities such as doctor's appointments, medication schedules, and dietary plans.

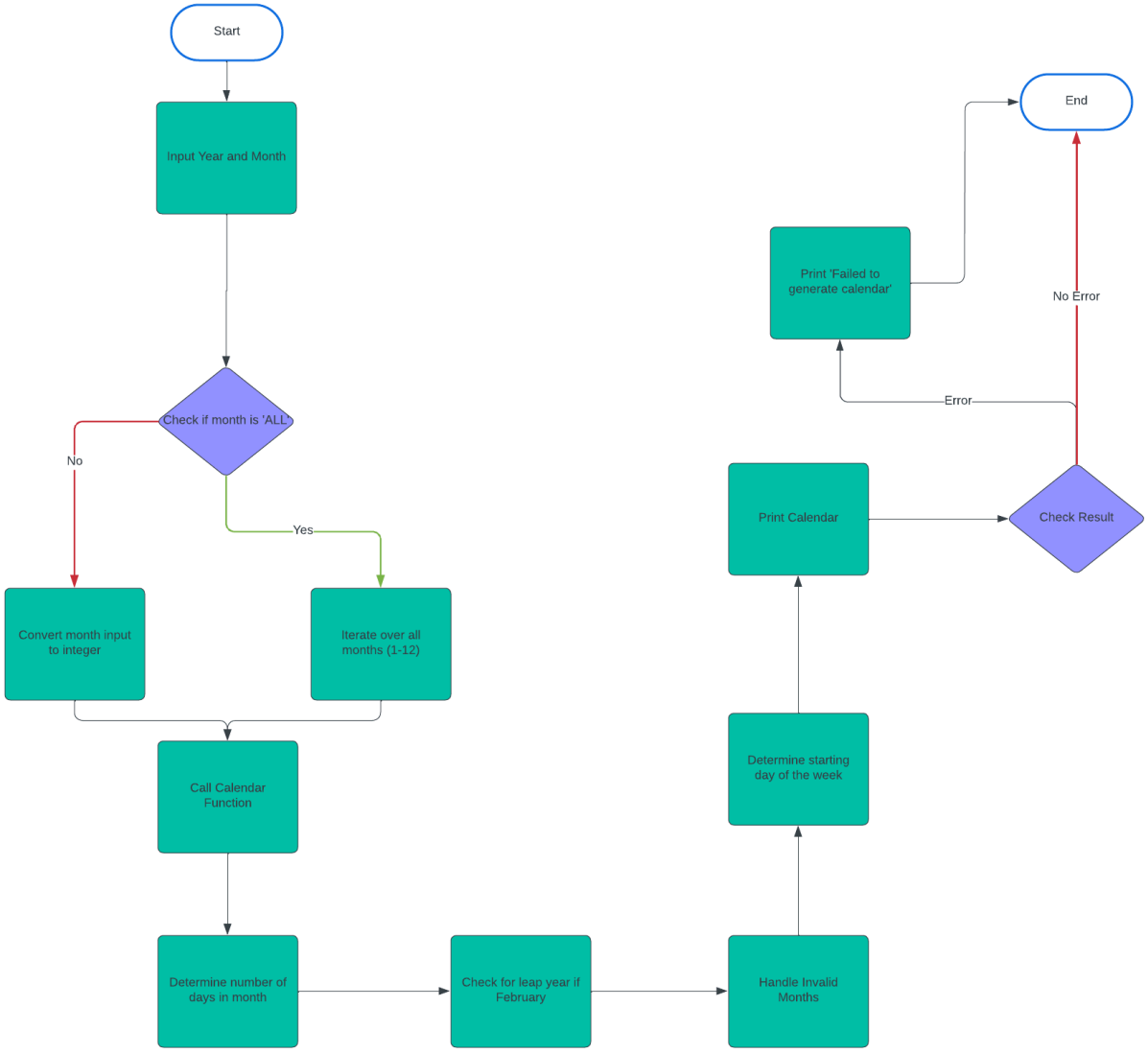
7. Social and Cultural Events: Users rely on the Gregorian calendar to stay informed about social and cultural events, such as holidays, festivals, concerts, sports events, and community gatherings. It helps them participate in and plan for various social activities throughout the year.
8. Long-Term Planning: The Gregorian calendar enables users to plan for long-term goals, milestones, and future events, such as weddings, graduations, retirement, and major life transitions. By visualizing their timeline, users can make strategic decisions and take proactive steps towards their aspirations.

PROCESS FLOW

1. Start: The program execution begins.
2. Input Year and Month:
    - The program prompts the user to input the desired year.
    - The user enters the year using the keyboard.
    - The program prompts the user to input the desired month (1-12) or "ALL" for all months.
    - The user enters the month using the keyboard.
3. Check Input:
    - If the user enters "ALL" for the month:
        - Proceed to step 4.
    - If the user enters a numeric value for the month:
        - Proceed to step 5.
    - If the user enters any other input:
        - Display "Invalid month" message.
        - End the program.
4. Generate Calendar for All Months:
    - Loop through each month (1 to 12):
        - Call the calendar function for the current year and month.
        - Display the calendar for the current month.

5. Generate Calendar for Specific Month:
    - Call the calendar function with the provided year and month.
    - The calendar function calculates the number of days in the month and the starting day of the week.
    - The calendar function prints the calendar for the specified month and year.
6. Display Calendar:
    - The calendar function prints the calendar layout, including the year and month header, days of the week, and dates.
    - Dates are printed in a grid format, with appropriate spacing and alignment.
    - The calendar is displayed in the console or terminal window.
7. End: The program execution ends.

This process flow outlines the steps involved in generating a calendar using the provided C code, including input handling, calendar generation, and out display.



TEST CASES

1. Leap Year Test Cases:
   -Input: Year 2020
    - Expected Output: Leap year (February has 29 days)
   - Input: Year 2100
    - Expected Output: Not a leap year (February has 28 days)

2. Month Length Test Cases:

- Input: Year 2023, Month 2 (February)
  - Expected Output: 28 days
- Input: Year 2023, Month 4 (April)
  - Expected Output: 30 days

3. Boundary Test Cases:
- Input: Year 1 (or any very early year)
  - Expected Output: Error or undefined behavior (handle edge case)
- Input: Year 9999 (or any very far future year)
  - Expected Output: Handle correctly without overflow or error

4. Invalid Input Test Cases:
- Input: Year -100 (negative year)
  - Expected Output: Error or undefined behavior (handle invalid input)
- Input: Month 13 (invalid month)
  - Expected Output: Error or undefined behavior

5.day of Week Test Cases:
- Input: Year 2024, Month 5, Day 14
  - Expected Output: Wednesday
- Input: Year 2023, Month 12, Day 25
  - Expected Output: Tuesday

7. Day Counter Test Cases:
- Input: Year 2023, Month 1, Day 1
  - Expected Output: Day of the year: 1
- Input: Year 2023, Month 12, Day 31
  - Expected Output: Day of the year: 365

8. Input- year: 2004 month-03
   Output -
   Enter year: 2004
   Enter month (1-12): 3

9. Enter year: 2024
   Enter month (1-12 or 'ALL' for all months): ALL
   Output : Calender of all the months of that year will be printed.

CODE :

```
#include <stdio.h>
#include <stdbool.h>

bool is_leap_year(int year) {
   if (year % 400 == 0)
      return true;
   else if (year % 100 == 0)
      return false;
   else if (year % 4 == 0)
      return true;
   else
      return false;
}
int days_in_month(int month, int year)
{
   if(month == 9 || month == 4 || month == 6 || month == 11)
   return 30;
   else if(month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 ||
month == 12)
   return 31;
   else if(month == 2 && is_leap_year(year)==true)
   return 29;
   else
   return 28;
}

int days_from_epoch(int day, int month, int year) {
   int days = 0;
   int i, j;
   int diff;

   if (year < 1970) {
      diff = 1970 - year;

      for (j = 1; j < diff; j++) {
         if (is_leap_year(1970 - j))
            days += 366; else
            days += 365;
      }
```

```
        for (i = month + 1; i <= 12; i++)
            days += days_in_month(i - 1, year);

        days += (days_in_month(month, year) - day + 1);
    } else {
        diff = year - 1970;

        for (j = 0; j < diff; j++) {
            if (is_leap_year(1970 + j))
                days += 366;
            else
                days += 365;
        }

        for (i = 1; i < month; i++)
            days += days_in_month(i, year);

        days += (day - 1);
    }

    return days;
}

int day_of_week(int day, int month, int year) {
    int _day = days_from_epoch(day, month, year) % 7;

    if (year > 1970) {
        switch (_day) {
            case 0:
                return 4;
            case 1:
                return 5;
            case 2:
                return 6;
            case 3:
                return 0;
            case 4:
                return 1;
            case 5:
```

```
            return 2;
        case 6:
            return 3;
      }
   } else {
      switch (_day) {
         case 0:
            return 4;
         case 1:
            return 3;
         case 2:
            return 2;
         case 3:
            return 1;
         case 4:
            return 0;
         case 5:
            return 6;
         case 6:
            return 5;
      }
   }
}

void display_month(int month, int year) {
   printf("\n      ");
   printf("Month %d, %d\n", month, year);

   printf(" Sun Mon Tue Wed Thu Fri Sat\n");

   int begin = day_of_week(1, month, year);
   int i, count = 0;
   for (i = 0; i < begin; i++) {
      printf("     ");
      ++count;
   }

   for (i = 1; i <= days_in_month(month, year); i++) {
      printf("%5d", i);
      count++;
```

```
        if (count % 7 == 0)
            printf("\n");
    }
    printf("\n\n");
}


int main() {
    int month;
    int year;

    printf("\nEnter Year = ");
    scanf("%d", &year);
    printf("Enter Month (1-12 or 'ALL') = ");
    if(scanf("%d", &month) == 1 && (month < 1 || month > 12)) {
        printf("\nInvalid Month, Please Enter Again (1-12) - ");
        return 1;
    }

    if (month == 0) {
        for (int m = 1; m <= 12; ++m) {
            display_month(m, year);
        }
    } else {
        while (year < 1) {
            printf("\nInvalid Year, Please Enter Again - ");
            scanf("%d", &year);
        }
        display_month(month, year);
    }

    return 0;
}
```

TEST RESULTS:

```
Enter Year = 2024
Enter Month (1-12 or 'ALL') = ALL

     Month 1, 2024
 Sun  Mon  Tue  Wed  Thu  Fri  Sat
       1    2    3    4    5    6
   7    8    9   10   11   12   13
  14   15   16   17   18   19   20
  21   22   23   24   25   26   27
  28   29   30   31


     Month 2, 2024
 Sun  Mon  Tue  Wed  Thu  Fri  Sat
                   1    2    3
   4    5    6    7    8    9   10
  11   12   13   14   15   16   17
  18   19   20   21   22   23   24
  25   26   27   28   29


     Month 3, 2024
 Sun  Mon  Tue  Wed  Thu  Fri  Sat
                        1    2
   3    4    5    6    7    8    9
  10   11   12   13   14   15   16
  17   18   19   20   21   22   23
  24   25   26   27   28   29   30
  31


     Month 4, 2024
 Sun  Mon  Tue  Wed  Thu  Fri  Sat
       1    2    3    4    5    6
   7    8    9   10   11   12   13
  14   15   16   17   18   19   20
  21   22   23   24   25   26   27
  28   29   30


     Month 5, 2024
 Sun  Mon  Tue  Wed  Thu  Fri  Sat
                 1    2    3    4
   5    6    7    8    9   10   11
  12   13   14   15   16   17   18
  19   20   21   22   23   24   25
  26   27   28   29   30   31
```

```
     Month 6, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                             1
  2    3    4    5    6    7    8
  9   10   11   12   13   14   15
 16   17   18   19   20   21   22
 23   24   25   26   27   28   29
 30


     Month 7, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
       1    2    3    4    5    6
  7    8    9   10   11   12   13
 14   15   16   17   18   19   20
 21   22   23   24   25   26   27
 28   29   30   31


     Month 8, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                  1    2    3
  4    5    6    7    8    9   10
 11   12   13   14   15   16   17
 18   19   20   21   22   23   24
 25   26   27   28   29   30   31



     Month 9, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
  1    2    3    4    5    6    7
  8    9   10   11   12   13   14
 15   16   17   18   19   20   21
 22   23   24   25   26   27   28
 29   30


     Month 10, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
            1    2    3    4    5
  6    7    8    9   10   11   12
 13   14   15   16   17   18   19
 20   21   22   23   24   25   26
 27   28   29   30   31
```

```
     Month 11, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                             1    2
  3    4    5    6    7    8    9
 10   11   12   13   14   15   16
 17   18   19   20   21   22   23
 24   25   26   27   28   29   30



     Month 12, 2024
Sun  Mon  Tue  Wed  Thu  Fri  Sat
  1    2    3    4    5    6    7
  8    9   10   11   12   13   14
 15   16   17   18   19   20   21
 22   23   24   25   26   27   28
 29   30   31
```