Design and Management of Departmental Calendar..

November 28, 2024

PROJECT BASED LEARNING

TITLE PROJECT: DESIGN AND MANAGEMENT OF DEPT. CALENDAR ..

GROUP ID :- 11

ATHARVA PARDESHI	**B24IT1056**
KARAN BHAPKAR	**B24IT1005**
OMKAR MARKAD	**B24IT1020**

- Dt of presentation: 29/11/2024

Objective of the Program: Develop a program in C that: Accepts a month and year as input. Displays the calendar for the given month. Handles leap years correctly. Demonstrate logical calculations for: Days in a month. The starting day of the week. Accepts a month and year as input. Displays the calendar for the given month. Handles leap years correctly. Demonstrate logical calculations for: Days in a month. The starting day of the week.

Key Features: Leap year detection. Calculation of the starting day of the month. Organized display of the calendar with proper alignment. Flexible to handle any year and month **input.**

Program Flow Diagram: Input: Accept year and month from the user.
Process:
Calculate the starting day of the year. Compute the starting day of the selected month. Determine the number of days in the month.
Output: Display the calendar in a formatted layout.
Diagram: Include a flowchart showing the steps (Input -> Calculation -> Output).

Leap Year Check: A year is a leap year if: It is divisible by 4. It is not divisible by 100 unless also divisible by 400.
Example:
2020: Leap year (divisible by 4).
1900: Not a leap year (divisible by 100, but not 400).
2000: Leap year (divisible by 400).

Code Explanation
Title: Code Walkthrough: Functions Used:
getDaysInMonth(): Determines the number of days in a given month.
getStartDayOfYear(): Calculates the starting day of the year.
printCalendar(): Displays the calendar.
Main Function:
Takes input from the user.
Calls helper functions to process and display results.

Sample Input/Output
Title: Program Demonstration
Content:

Input Example:

Month: 11
Year: 2024
Output Example:

markdown
Copy code

```
   November 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
```

Challenges Faced
Title: Challenges During Development
Content: Accurate calculation of starting day for any year. Handling special cases like leap years . Proper alignment of days in the calendar display.

Learning Outcomes
Title: What We Learned
Content: Writing modular code with functions. Handling complex date calculations. Implementing logical and conditional operations in C. Formatting output for better user experience.

Title: Conclusion
Content: The program successfully generates calendars for any valid month and year. Demonstrates the importance of logic in programming. Can be further enhanced for additional features like event marking or graphical UI. Closing Note:
"Calendars remind us of the passage of time, and this program demonstrates how simple logic can make a powerful utility."

Title: Questions and Answers
Content: Invite questions and feedback from the audience.
Presentation Tips:
Use visuals like flowcharts or screenshots of the program output to make your presentation engaging.
Avoid overloading slides with text; use bullet points.
Speak confidently and explain key sections like leap year logic and the code structure in detail.
Let me know if you'd like a tailored design or more details for any slide!
How to Create a Calendar Program in C?
Calendars play a vital role in organizing our schedules and tracking important dates. In this blog, we will explore how to create a simple calendar program in C that can display the calendar for any month and year provided by the user. This program demonstrates key programming concepts such as leap year calculations, modular arithmetic, and formatted output.

Objective: The objective of this program is to: Take a month and year as input. Calculate the starting day of the week for the given month. Display the calendar in a neatly formatted

layout. Key Concepts Behind the Calendar. To build a calendar program, we need to understand some essential concepts:

1. Leap Year Check

A leap year affects the number of days in February (29 days instead of 28). The rules are:

A year is a leap year if: It is divisible by 4 but not by 100, or It is divisible by 400.

For example:

2020: Leap year (divisible by 4).

1900: Not a leap year (divisible by 100 but not 400).

2000: Leap year (divisible by 400).

2. Calculating the Starting Day

To align the calendar correctly, we calculate the weekday of the first day of the given month. Days are represented numerically: Sunday = 0, Monday = 1, ..., Saturday = 6.

Using modular arithmetic, we can calculate the starting day of any year and month.

3. Days in Each Month: Each month has a fixed number of days except February, which depends on whether it is a leap year:

January, March, May, July, August, October, December = 31 days.

April, June, September, November = 30 days.

February = 28 or 29 days.

```
#include <stdio.h>
// Function to get the number of days in a month
int getDaysInMonth(int month, int year) {
    if (month == 2) {
        return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)) ? 29 : 28;
    }
    return (month == 4 || month == 6 || month == 9 || month == 11) ? 30 : 31;
}

// Function to calculate the starting day of a given year
int getStartDayOfYear(int year) {
    int day = 1; // January 1, year 1 is Monday
    for (int y = 1; y < year; y++) {
        day += (y % 4 == 0 && (y % 100 != 0 || y % 400 == 0)) ? 366 : 365;
    }
    return day % 7; // Day of the week
```

```
}
// Function to print the calendar for a given month and year
void printCalendar(int month, int year) {
    char *monthNames[] = {"January", "February", "March", "April", "May", "June",
                          "July", "August", "September", "October", "November", "December"};
    char *weekDays[] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
    int daysInMonth = getDaysInMonth(month, year);
    int startDay = getStartDayOfYear(year);

    for (int i = 1; i < month; i++) {
        startDay = (startDay + getDaysInMonth(i, year)) % 7;
    }

    // Print month and year
    printf("\n   %s %d\n", monthNames[month - 1], year);
    for (int i = 0; i < 7; i++) {
        printf("%s ", weekDays[i]);
    }
    printf("\n");

    // Print spaces before the first day
    for (int i = 0; i < startDay; i++) {
        printf("    ");
    }

    // Print the days of the month
    for (int day = 1; day <= daysInMonth; day++) {
        printf("%3d ", day);
        if ((startDay + day) % 7 == 0) {
            printf("\n");
        }
    }
    printf("\n");
}

int main() {
```

```
    int month, year;

    // User input
    printf("Enter month (1-12): ");
    scanf("%d", &month);
    printf("Enter year: ");
    scanf("%d", &year);

    // Validate input
    if (month < 1 || month > 12 || year < 1) {
        printf("Invalid input. Please enter a valid month (1-12) and year (>0).\n");
        return 1;
    }

    // Display the calendar
    printCalendar(month, year);

    return 0;
}
```

How It Works?

Leap Year Logic: The getDaysInMonth function determines the number of days in February (28 or 29) based on leap year conditions.

Starting Day of the Month: Using getStartDayOfYear, the program calculates the weekday of January 1st for the given year. It adjusts the starting day for subsequent months.

Calendar Layout: The printCalendar function displays the days of the week as headers and aligns the dates accordingly.

Input:

mathematica

```
Enter month (1-12): 11
Enter year: 2024
```

Output:

markdown

Copy code

November 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

Challenges Faced: Calculating the starting day of any year and month accurately. Aligning the dates correctly under their respective weekdays. Handling edge cases for leap years. Potential Enhancements While this program is functional, it can be improved in the following ways:

Add navigation for users to view previous or next months. Allow users to add events or notes for specific dates. Create a graphical interface for better usability.

Conclusion:

This calendar program is a great example of solving real-world problems with programming logic. By breaking down the problem into manageable steps, we were able to:

Implement leap year checks.

Align dates under correct weekdays.

Display a clear and formatted calendar.

DEPARTMENTAL CALENDAR

Visit profile

Archive

November 20241

Report Abuse