

Four hundred years calendar

Assignment #02



Muhammad Qasim Ali 4392-FOC-BSSE/F22A

Muhammad Rehan Nawaz 4390-FOC-BSSE/F22A

Aaban Rehman 4375-FOC-BSSE/F22A

Siam Khan 4333-FBAS-BSSE/F21/A

Software construction and development

Sir Shakir rasheed

28th May, 2025

## 400 Years Calendar Generator Using C#

## 1. Project Overview

This project focuses on the development of a console-based application that generates and displays monthly calendars across a 400-year range from the current year (2025) to the previous 400 years (i.e., up to the year 1625).The application uses the Gregorian calendar system and accounts for leap years, correct weekday alignment, and the varying number of days per month.

The objective is to gain hands-on experience with date/time manipulation, console formatting, nested iteration, and collaborative development using Git and GitHub.

## 2. Team Members and Collaboration Strategy

### Team Composition

* The team consists of four members from the BSSE program.
* All members actively contributed to the development, testing, formatting, and documentation of the application.

### Collaboration Approach

* A GitHub repository was created to host the codebase and manage version control.
* Each member cloned the repository, created branches or committed directly, and pushed their contributions.
* Git was used for tracking changes, with commits made for each major task or update.
* Screenshots has been provided, showing screenshots of Git logs with each member's contributions.

## 3. Objectives

* Develop a functional calendar application in C#.
* Automatically generate and display monthly calendars from 1625 to 2025.
* Ensure accurate weekday alignment and leap year handling.
* Demonstrate collaborative development using Git.
* Provide clear and well-formatted calendar output via the console.

## 4. Tools and Technologies

* **Programming Language:** C#
* **Framework:** .NET 8.0 SDK
* **IDE:** Microsoft Visual Studio 2022
* **Version Control:** Git
* **Repository Hosting:** GitHub
* **Documentation Tool:** Microsoft Word

## 5. Functional Requirements

* The application must generate calendars for each month between the years 1625 and 2025.
* Each calendar should:  
  + Display the correct name of the month and year.
  + Include a header row with weekday abbreviations (Su, Mo, Tu, etc.).
  + Correctly align day numbers under the appropriate weekday columns.
  + Accurately determine the number of days in each month using built-in date functions.
  + Account for leap years (including century rules: divisible by 4, except those divisible by 100 unless also divisible by 400).

## ****6. System Design****

### ****Menu Design:****

The application begins by displaying a menu with the following options:

1. **Print Full Calendar (1625–2025)**
   * Displays the complete calendar for all 400 years from 1625 to 2025 at once.
2. **Print Calendar for a Specific Year**
   * Asks the user to input a year between **1625 and 2025**.
   * Displays the calendar for that specific year.
   * After displaying, it prompts the user to:
     + Enter another year to view, or
     + Type **exit** to return to the main menu.
3. **Exit Program**
   * Ends the application.

### ****Input****

* The program requires the user to select a menu option (1, 2, or 3).
* In case 2, it prompts the user to enter a year between **1625 and 2025**.
* The user may enter **exit** at any point during the year-specific view to return to the main menu.

### ****Output****

* If the user selects **option 1**, the application prints calendars for **4800 months** (400 years × 12 months).
* If the user selects **option 2**, the application displays only the calendar of the specified year.
* All output is shown in a **structured and readable format in the console**, with proper spacing and month-wise arrangement.

### Processing Logic

* The application uses the DateTime class to determine:  
  + The first day of each month.
  + The total number of days in each month.
* Nested loops iterate over years and months.
* Proper spacing is applied to align dates under the correct weekday headers.

## 7. Challenges Faced

* **Calendar Alignment:** Ensuring proper spacing so that each date appears under the correct weekday required adjustments and testing.
* **Leap Year Accuracy:** Leap year rules (especially centennial exceptions) were carefully validated using DateTime methods.
* **Formatting Uniformity:** Maintaining a consistent output format for different month lengths and starting weekdays took additional logic.
* **Team Coordination:** Synchronizing commits and resolving minor merge conflicts while using Git collaboratively was part of the learning curve.

## 8. Version Control and Collaboration Documentation

Each member used GitHub to collaboratively manage source code. The Git log history clearly shows multiple commits by different authors, representing team-based development.

### GitHub Repository

* Repository created and shared with all group members.
* Each member pulled, edited, committed, and pushed changes during different phases of the project.

### Git Log Documentation

Screenshots are attached with this report, which includes:

* Visual proof of contributions using git log screenshots.
* Dates, commit messages, and author names confirming individual work.

**[Git Log Screenshot Placeholder – Insert into Word Document]**

## 9. Outcomes

* A fully functional calendar generation application was developed.
* The system can generate calendars from 2000 to 2400 accurately and display them in a clean layout.
* Team members gained practical experience in:  
  + C# and .NET Console Programming
  + Date and Time manipulation using DateTime
  + Console UI formatting
  + Git and GitHub for team-based version control
* All learning objectives related to the Software Construction and Development course were met.

## 10. Conclusion

The 400 Years Calendar Generator project fulfilled all outlined requirements. It demonstrates the team’s capability to apply core software development concepts such as control flow, data structures, modular programming, and source control collaboration.

The use of C#’s built-in libraries enabled reliable and accurate calendar calculations. Collaborative development practices ensured high-quality code and documentation. The project stands as a strong example of planning, division of tasks, technical implementation, and teamwork.

## 