**ARTIFICIAL INTELLIGENCE DEPARTMENT**

# Total Marks:

**Obtained Marks:**

**PROJECT NO#2**

**REPORT**

Object Oriented Programming Techniques (LAB)

**CALENDAR APPLICATION**

# Submitted To: Sir Hamza Imran

**Student Name:** **M.Muneeb, Muddasir, Adnan**

**Section Name:** **BSAI-2A**

# Reg. Number: 22108190, 22108251,22108166

**ARTIFICIAL INTELLIGENCE DEPARTMENT**

**INTRODUCTION: -**

In today's fast-paced world, staying organized and keeping track of your schedule is more important than ever. Our calendar application is designed to be your ultimate time-management companion. Whether you're a busy professional, a student juggling assignments, or someone trying to manage a hectic personal life, our app is here to help. Designing a calendar application using Object-Oriented Programming (OOP) principles in C++ involves creating classes that encapsulate the various functionalities of a calendar. OOP allows for a modular design, where different aspects of the calendar are managed by separate, interacting classes.

**PROCEDURE: -**

The procedure for a calendar application, particularly focusing on a C++ implementation using Object-Oriented Programming, involves several steps. This procedure outlines how different components of the application interact with each other to provide functionality. Here's a simplified version of the procedure:

### **Initialization**

* Start the application.
* Initialize the main *Calendar* object.
* Load existing events from a file/database (if persistence is implemented).

### **2. User Interface Interaction**

* Display the main menu or interface to the user.
* Provide options like viewing the calendar, adding an event, editing an event, or deleting an event.

### **3. Handling User Input**

* When the user selects an option, prompt for necessary details. For example, if adding an event, ask for the title, date, time, etc.
* Validate user inputs.

### **4. Processing Requests**

* Based on the user's choice, execute the corresponding functionality:
  + ***View Calendar***: Display events in a specified format (daily, weekly, monthly).
  + ***Add Event***: Create a new *Event* object and add it to the *Calendar.*
  + ***Edit Event***: Find the event in the *Calendar* and modify its details.
  + ***Delete Event***: Remove the event from the *Calendar.*

### **5. Updating the Calendar**

* After any add, edit, or delete operation, update the calendar's event list.
* If persistence is implemented, save changes to the file/database.

### **6. Displaying Results**

* Show the updated calendar or confirmation of actions (like successful addition or deletion of an event).
* For viewing events, format the display to show event details.

### **7. Loop Back or Exit**

* After completing an action, offer the user to perform another action or exit the application.
* If exiting, perform any necessary cleanup, like saving data.

### **8. Closing the Application**

* Save any unsaved changes.
* Close any open resources (files, database connections).
* Exit the program.

### **Additional Features and Considerations**

* ***Error Handling***: Implement robust error handling for user inputs, file operations, etc.
* ***Notifications/Reminders***: Implement a system to alert users of upcoming events.
* ***User Authentication (Optional)****:* If the calendar is user-specific, implement a login system.

**OBJECTIVE: -**

The primary objective of a calendar application is to provide an efficient and user-friendly platform for managing time and scheduling. This encompasses the ability to create, view, and edit events and appointments, often with varying levels of detail such as title, location, start and end times, and descriptions. An effective calendar application not only serves as a simple tool for remembering dates but also acts as a personal organizer, helping users plan their daily, weekly, and monthly activities. It often includes features like reminders and notifications to ensure users are aware of upcoming events or deadlines, enhancing productivity and time management. Additionally, in more advanced implementations, the calendar application may offer synchronization with other calendar services, sharing capabilities for collaborative planning, and support for recurring events, thereby catering to a broad range of personal and professional scheduling needs. The ultimate goal is to provide a seamless and intuitive interface that helps users stay organized and efficient in their day-to-day lives.

**SCOPE OF CALENDAR APPLICATION: -**

The scope of a calendar application, especially in the context of development using Object-Oriented Programming (OOP) in C++, can encompass a wide range of features and functionalities. Here's an overview of the potential scope:

* **Date and Time Management**: Ability to represent and manipulate dates and times effectively.
* **Event Creation and Management**: Users can create, edit, view, and delete events.
* **Multiple Views**: Displaying the calendar in different formats like day, week, month, and year views.
* **Recurring Events**: Support for events that recur daily, weekly, monthly, or annually.
* **Reminders and Notifications**: Setting up alerts for upcoming events.
* **Time Zone Support**: Managing events across different time zones.
* **User Interface**: A graphical user interface (GUI) for easier interaction, if not command-line based.
* **User Accounts**: Multiple users with personal calendars, requiring authentication and authorization.
* **External Calendar Integration**: Syncing with other calendar services like Google Calendar or Outlook.
* **Sharing and Collaboration**: Allowing users to share events or entire calendars with others.
* **Database Integration**: Storing events and user information in a database for persistence.
* **Exporting and Importing Data**: Ability to export calendar data to formats like iCal, CSV, etc., and import from them.
* **Customizable Views**: Allowing users to customize how they view the calendar.
* **Event Categories and Tags**: Categorizing events for better organization.
* **Accessibility Features**: Making the application accessible to users with disabilities.
* **Localization**: Adapting the application for different languages and cultures.
* **Cross-Platform Compatibility**: Developing versions for web and mobile platforms.
* **API Development**: Creating an API for other applications to interact with the calendar.
* **Data Security**: Ensuring the privacy and security of user data.
* **Regular Updates and Maintenance**: To address security vulnerabilities and bugs.
* **Event Analytics**: Insights on user's calendar usage patterns.
* **Reporting Tools**: Generating reports for users' activities and events.

**FUTURE SCOPE OF CALENDAR APPLICATION: -**

The future scope of calendar applications is expansive, considering the rapid advancements in technology and changing user needs. Here are some potential areas of development:

1. **Integration with Artificial Intelligence (AI)**: AI can be used to enhance the functionality of calendar applications in various ways. This could include smart scheduling where the app suggests the best times for meetings based on participants' habits, automatic event creation from emails, and predictive analytics to help in better time management.
2. **Advanced Customization and Personalization**: Future calendar applications could offer more personalized experiences, adapting to individual user preferences and routines. This might include customizable views, event suggestions based on past activities, and adaptive reminder systems.
3. **Improved Natural Language Processing (NLP)**: Enhanced NLP capabilities could allow users to interact with the calendar application more naturally, using conversational language to create, modify, or query events.
4. **Augmented Reality (AR) and Virtual Reality (VR) Integration**: Implementing AR and VR could transform how we view and interact with calendars. For example, using AR to overlay a calendar on a real-world view or VR to engage in a fully immersive scheduling environment.
5. **Internet of Things (IoT) Connectivity**: Calendars could be integrated with various IoT devices. For example, your alarm clock could sync with your calendar to wake you up earlier if you have an early meeting, or your smart home could prepare for a scheduled event by adjusting lighting and temperature.
6. **Enhanced Collaboration Tools**: Future calendar applications might offer more robust tools for collaboration, making it easier to schedule and manage events involving multiple parties, shared task lists, and collaborative goal tracking.
7. **Health and Wellness Integration**: Calendars could incorporate health and wellness tracking, suggesting breaks, exercise sessions, or meditation based on your schedule. This integration could be particularly beneficial for users with a busy schedule, helping to maintain a balance between work and personal well-being.
8. **Cross-Platform Synchronization and Functionality**: Improved synchronization across various platforms and devices, ensuring seamless experience whether the user is on their phone, computer, tablet, or a wearable device.
9. **Data Security and Privacy Enhancements**: As calendars hold sensitive personal and professional information, future developments will likely focus more on securing this data and ensuring user privacy.
10. **Eco-friendly and Socially Responsible Features**: Features encouraging sustainable and socially responsible behaviors, like suggesting public transport options for meetings, could become more prevalent.

**FEATURES OF CALENDAR APPLICATION: -**

This C++ program implements a calendar application with several notable features and design choices. Here's a breakdown of its key aspects:

1. **Object-Oriented Design**:

* The program uses classes (*Calendar,* *LeapYearCalendar,* and *FileCalendar*) to encapsulate the functionalities, following the principles of Object-Oriented Programming (OOP).

1. **Base Class - Calendar**:

* Serves as the base class with common attributes and functionalities.
* Includes methods for leap year checking, current date display, calculating the number of days in a month, getting month names, and determining the day number.
* Uses a pure virtual function *displayCalendar()* to enforce that derived classes implement this method.

1. **Leap Year Support**:

* The *LeapYearCalendar* class, derived from *Calendar*, specifically handles leap years, ensuring accurate calendar representation for such years.

1. **Calendar Display**:

* The *displayCalendar* method in *LeapYearCalendar* prints out a formatted calendar for the entire year, correctly aligning days of the week.

1. **File Saving Capability**:

* The *FileCalendar* class, inheriting from *LeapYearCalendar*, adds functionality to save the displayed calendar to a file. This is useful for record-keeping or printing purposes.

1. **Friend Function - Greeting**:

* Implements a friend function *greeting*, demonstrating the use of friend functions in C++. This function accesses protected members of the *Calendar* class.

1. **Static Function for Current Date**:

* The static method *displayCurrentDate* in the *Calendar* class displays the current system date, showcasing the use of static methods.

1. **Interactive User Interface**:

* The main function provides an interactive console-based interface, allowing users to input a year, view the calendar for that year, and choose to save the calendar to a file.

1. **Leap Year Verification**:

* When a year is entered, the program informs the user whether it is a leap year or not.

1. **File I/O**:

* The program demonstrates file input/output operations in C++, as it writes the calendar to a file if the user chooses to do so.

1. **Error Handling**:

* Basic error handling is included for file operations, ensuring the program notifies the user if a file cannot be opened for writing.

1. **Program Flow Control**:

* The program uses a loop to allow users to continuously input new year’s or exit the application, demonstrating control flow management.

This program is a comprehensive example of applying OOP concepts in C++, with a focus on calendar-related functionalities. It combines fundamental programming constructs with more advanced features like class inheritance, polymorphism (through the pure virtual function), file I/O, and static methods.

**DISADVANTAGES OF CALENDAR APPLICATION: -**

The calendar application you've written in C++ demonstrates solid programming skills and a good understanding of Object-Oriented Programming (OOP) principles. However, like any software, there are potential disadvantages or areas for improvement:

1. **Platform Dependency**:

* C++ is a platform-dependent language. This means the same code might not work on different operating systems without modifications. Portability can be an issue.

1. **Lack of User Interface**:

* The application operates in a console environment. It lacks a graphical user interface (GUI), which might not be user-friendly for all users, especially those who are not comfortable with command-line interfaces.

1. **Limited Functionality**:

* The application provides basic calendar functionalities. However, it lacks more advanced features like event reminders, integration with other calendar services, or the ability to handle time zones.

1. **File-Based Data Management**:

* Data (like saved calendars) is managed through files. This approach is less efficient and scalable compared to using a database. Also, there's no functionality for data retrieval once saved.

1. **No Networking Capabilities**:

* The application does not support networking features like syncing with other devices or sharing calendars with others.

1. **Manual Input and Error Handling**:

* The application relies heavily on user input without robust error handling. This could lead to crashes or incorrect outputs if the input is not as expected.

1. **No Multilingual Support**:

* The application is likely designed for English speakers only, lacking internationalization and localization support for different languages and cultures.

1. **No Real-Time Updates or Notifications**:

* The application does not provide real-time updates or notifications, which are essential features in modern calendar applications.

1. **Lack of Accessibility Features**:

* The application does not include accessibility features for users with disabilities, such as screen reader compatibility or high-contrast display options.

1. **Memory Management**:

* Being a C++ application, it requires careful memory management. Although modern C++ has smart pointers and other features, improper handling can still lead to memory leaks or inefficient memory usage.

**TOPICS RELATED TO OOPs: -**

1. **Encapsulation**: The concept of bundling data and methods that operate on that data within a single unit or object.
2. **Inheritance**: The mechanism where a new class (derived class) inherits attributes and behaviors (methods) from an existing class (base class).
3. **Polymorphism**: The ability of different objects to respond to the same method call in a way that is specific to their individual types.
4. **Abstraction**: Hiding the complex reality while exposing only the essential features of an object.

### **Classes**

1. **Constructor**: Special method in a class used for initializing new objects.
2. **Destructor**: Method that is automatically invoked when an object is destroyed to clean up resources.
3. **Class Variables**: Variables that are shared across all instances of a class.
4. **Instance Variables**: Variables that are unique to each instance of a class.
5. **Static Methods**: Methods that belong to the class rather than any particular object instance.
6. **Class Methods**: Methods that are bound to the class and not the instance of the class.
7. **Overloading**: The ability to define the same method multiple times with different parameters.
8. **Overriding**: The ability of a subclass to provide its own implementation for a method that is already defined in its superclass.

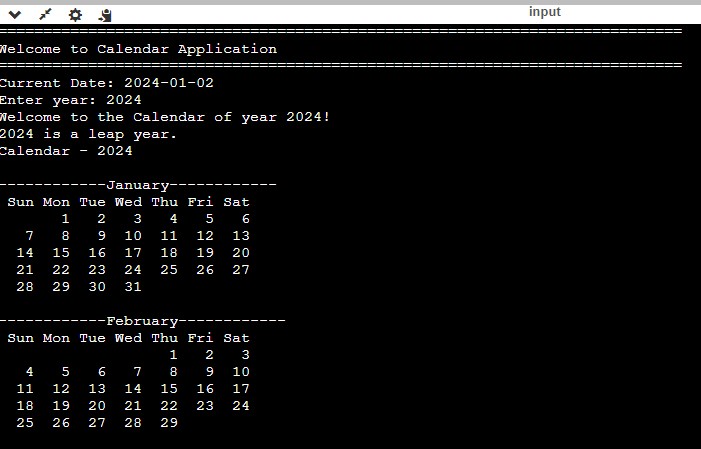
### **Structures**

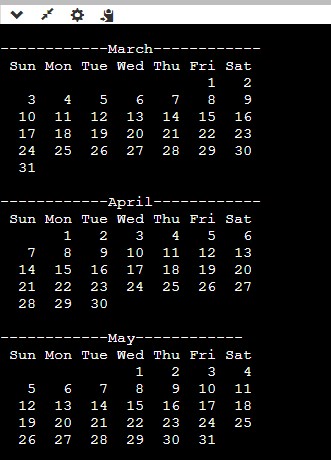
1. **Declaration**: Defining a new structure type with keyword **struct** (C, C++) or **class** with public access (C++).
2. **Initialization**: Creating instances of the structure.
3. **Nested Structures**: Structures within structures.
4. **Arrays of Structures**: Using arrays to store multiple structure elements.
5. **Structure Padding**: Compiler automatically adds padding to ensure memory alignment.

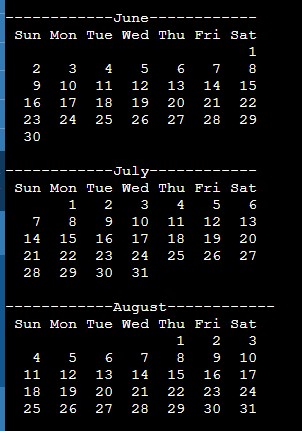
### **Functions**

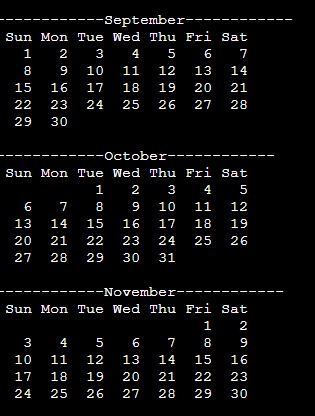
1. **Function Declaration**: Specifies a function's name, return type, and parameters.
2. **Function Definition**: Provides the actual body of the function.
3. **Function Overloading**: Defining multiple functions with the same name but different parameters.
4. **Function Overriding**: Providing a new implementation for a function in a subclass.
5. **Recursion**: Functions calling themselves.
6. **Lambda Functions**: Anonymous functions often used for short, throwaway functionalities.
7. **Variadic Functions**: Functions that accept an indefinite number of arguments.
8. **Pure Functions**: Functions where the output is determined only by its input, without any observable side effects.

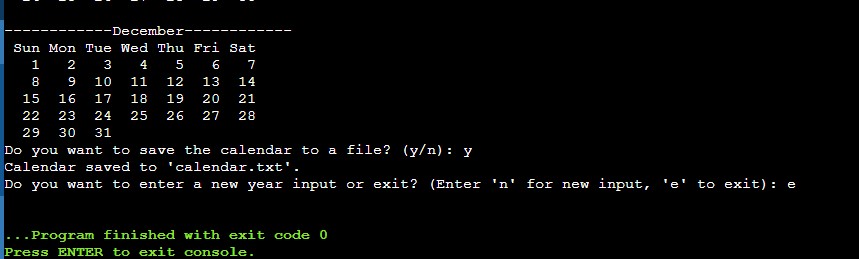
**OUTPUT: -**

****

****

****

****

****

**EXPLANATION: -**

This C++ program is a calendar application that demonstrates various aspects of object-oriented programming, including inheritance and friend functions. I'll explain each section in points:

### **Include Directives and Namespace**

* ***#include directives:*** These lines include standard library headers necessary for the program.
* ***using namespace std;*** : This line allows the program to use standard library names without the *std::* prefix.

### **Base Class: Calendar**

1. ***Class Definition***: *class Calendar* defines a base class for calendars.
2. ***Protected Attribute***: *int year;* holds the year for the calendar.
3. ***Constructor***: *Calendar(int y) : year(y) {}* initializes a Calendar object with a given year.
4. ***Leap Year Check***: *bool isLeapYear()* checks if the current year is a leap year.
5. ***Display Current Date***: *static void displayCurrentDate()* displays the current system date.
6. ***Number of Days in a Month***: *int numberOfDays(int monthNumber)* returns the number of days in a given month.
7. ***Get Month Name***: *string getMonthName(int monthNumber)* returns the name of a month.
8. ***Calculate Day Number***: *int dayNumber(int day, int month, int year)* calculates the day of the week for a given date.
9. ***Pure Virtual Function***: *virtual void displayCalendar() = 0;* is a pure virtual function for displaying the calendar.
10. ***Friend Function***: *friend void greeting(Calendar &cal);* declares a friend function for the class.

### **Derived Class: LeapYearCalendar**

1. ***Inheritance***: Inherits from the *Calendar* class.
2. ***Constructor***: Initializes the base class *Calendar*.
3. ***Override displayCalendar***: Overrides the *displayCalendar* function to display the calendar for the year.

### **Derived Class: FileCalendar**

1. ***Inheritance***: Inherits from *LeapYearCalendar*.
2. ***Constructor***: Calls the constructor of *LeapYearCalendar*.
3. ***Save to File***: *void saveToFile(const string &filename)* saves the calendar to a file.

### **Friend Function: greeting**

* ***Greeting Function***: *void greeting(Calendar &cal)* displays a greeting message. It has access to the protected members of *Calendar*.

### **Main Function**

1. ***Infinite Loop***: A loop to keep the program running until the user decides to exit.
2. ***Welcome Message***: Displays a welcome message.
3. ***Display Current Date***: Calls *Calendar::displayCurrentDate()* to display the current date.
4. ***Year Input***: Prompts the user to enter a year.
5. ***Create Calendar Object***: Creates a *FileCalendar* object for the entered year.
6. ***Greet User***: Calls the *greeting* function.
7. ***Leap Year Check***: Displays if the entered year is a leap year.
8. ***Display Calendar***: Calls *displayCalendar* to display the calendar.
9. ***Save Option***: Asks the user if they want to save the calendar to a file.
10. ***Save to File***: If the user chooses to save, the calendar is saved to "calendar.txt".
11. ***Continue Option***: Asks the user if they want to process a new year or exit.
12. ***Exit Condition***: Exits the loop and ends the program if the user chooses to exit.

***\*\*The End\*\****