Venue & Event Management System-Assignment 2

Prakrititz Borah (IMT2023547)

Overview

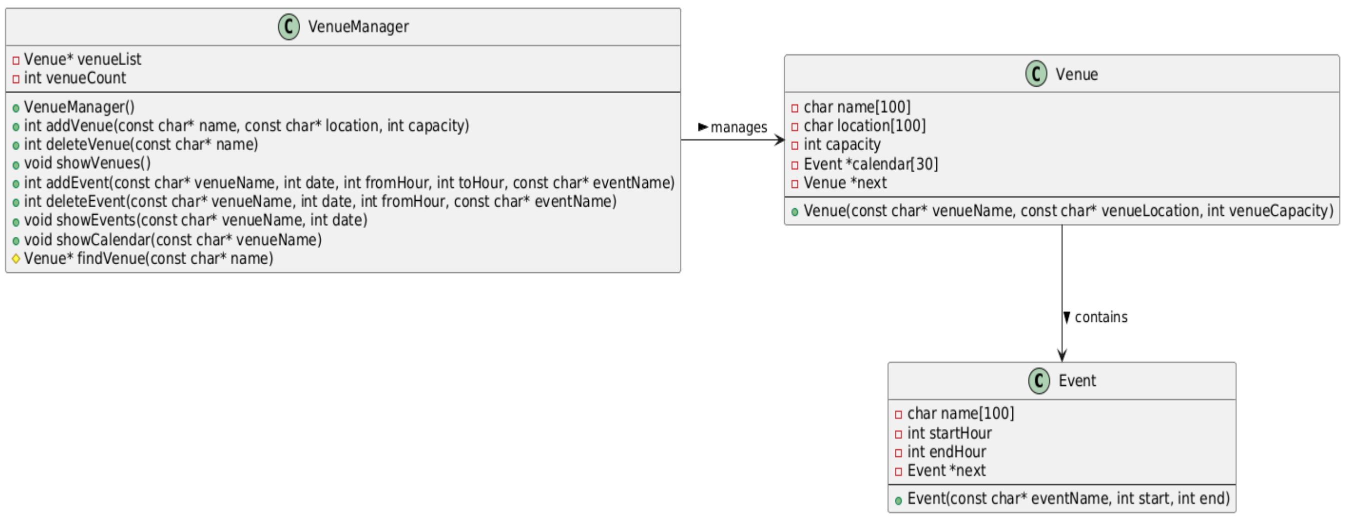
This C++ program implements a basic venue and event management system. It allows users to add, delete, and view venues and events. The system manages a collection of venues, where each venue can have multiple events scheduled on different days. The program is designed to handle up to 100 venues and maintain a calendar of events for up to 30 days per venue.

Code Structure

The code structure is designed to be modular, ensuring clarity and maintainability. Below is an outline of the main components:

1. **Class Definitions:**
   - **`class Event { ... }`**:
     Defines the structure for an event, including attributes such as event name, date, time, etc. Contains member functions for event management.
   - **`class Venue { ... }`**:
     Defines the structure for a venue, which includes a list of events. Contains member functions to manage venue details and the associated events.
   - **`class VenueManagementSystem { ... }`**:
     Manages the overall system, including a list of venues. Contains functions to add, delete, and display venues and events.
2. **Main Function:**
   - **`int main() { ... }`**:
     The entry point of the program. It initializes the VenueManagementSystem object and provides a menu-driven interface for users to interact with the system.
3. **Function Definitions:**
   - **`void Venue::addEvent(...)`**:
     Adds a new event to the venue, checking for conflicts and validating inputs.**`void Venue::deleteEvent(...)`**:
     Deletes an event from the venue after verifying its existence.
   - **`void VenueManagementSystem::addVenue(...)`**:
     Adds a new venue to the system.
   - **`void VenueManagementSystem::deleteVenue(...)`**:
     Deletes a venue from the system after ensuring no events are scheduled.
   - **`void VenueManagementSystem::displayVenues(...)`**:
     Displays all venues along with their scheduled events.

4. Uml Structure:


VenueManager
Venue* venueList
int venueCount
VenueManager()
int addVenue(const char* name, const char* location, int capacity)
int deleteVenue(const char* name)
void showVenues()
int addEvent(const char* venueName, int date, int fromHour, int toHour, const char* eventName)
int deleteEvent(const char* venueName, int date, int fromHour, const char* eventName)
void showEvents(const char* venueName, int date)
void showCalendar(const char* venueName)
Venue* findVenue(const char* name)
manages
Venue
char name[100]
char location[100]
int capacity
Event *calendar[30]
Venue *next
Venue(const char* venueName, const char* venueLocation, int venueCapacity)
contains
Event
char name[100]
int startHour
int endHour
Event *next
Event(const char* eventName, int start, int end)

Design Decisions

Why Linked List Structure?

In this program, a linked list structure is employed for managing venues and events. The key reasons for this choice include:

- **Dynamic Memory Management:**
  Linked lists enable efficient memory usage by dynamically creating nodes based on the number of venues and events. This flexibility eliminates the need to predefine the maximum number of venues or events, making the program adaptable to varying sizes of data.
- **Ease of Insertion/Deletion:**
  Linked lists facilitate efficient insertion and deletion of nodes, particularly when handling sequential data. In this program, it allows for seamless addition and removal of venues and events without requiring memory shifts, which would be necessary if arrays were used.
- **Scalability:**
  The linked list structure inherently scales with the data size. As the number of venues and events increases, the structure remains efficient without necessitating major changes to the codebase.

Error Handling

To ensure robustness and provide a user-friendly experience, comprehensive error handling is integrated throughout the program. The key aspects of error handling include:

- **Memory Allocation Failures:**
  During dynamic memory allocation for new venues or events, the program checks whether the allocation was successful. If not, it gracefully exits the operation and prints an error message to inform the user.
- **Validation of Inputs:**
  The program rigorously validates all inputs, including dates, times, and venue names,

before processing them. For instance, it ensures that the specified date and time are within valid ranges and verifies the existence of a venue or event before attempting any operations on them.

- **Conflict Detection:**
  When adding events, the program checks for any time conflicts with existing events at the venue. If a conflict is detected, the program returns an error and does not add the event, preserving the integrity of the schedule.
- **User Feedback:**
  In the event of errors, the program provides informative feedback, specifying the nature of the error (e.g., "Venue doesn’t exist" or "Invalid input"). This feedback helps users understand the issue and guides them on how to correct it.

Memory Management

The program employs dynamic memory allocation for managing venues and events. To prevent memory leaks, all allocated memory is properly freed before the program exits, ensuring efficient use of system resources.