Table of contents

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | **page** |
| **1.** | | **Algorithm** | [**3-4**](#Algorithm) |
| **2.** | | **Flowchart** | [**5-13**](#flowchart) |
| **3.** | | **Coding** | [**14-20**](#Coding) |
| **4.**  **5.** | | **Test data**  **Output** | [**21-27**](#TestData)  [**28-31**](#Output) |
|  |  | |  |

Algorithm

**STEP 1**: Start

**STEP 2**: Display the main menu with available options

**STEP 3**: Read the user's choice

**STEP 4**: If the user chooses to add an event (option == 1)

**STEP 4 (a)**: Prompt the user to enter event details (name, date, time, location, description [optional])

**STEP 4 (b)**: Read and validate input (correct date/time format, non-empty fields)

**STEP 4 (c)**: Check if an event with the same name already exists.

**STEP 4 (d)**: If no conflict, store the event

**STEP 4 (e)**: Display a success message

**STEP 5**: If the user chooses to update an event (option == 2)

**STEP 5 (a)**: Prompt the user to enter the name of the event to update

**STEP 5 (b)**: Read event name and verify its existence

**STEP 5 (c)**: Display a list of updatable fields (name, date, time, location, description)

**STEP 5 (d)**: Prompt the user to select a field and enter a new value

**STEP 5 (e)**: Validate and update the event details

**STEP 5 (f)**: Display a success message

**STEP 6**: If the user chooses to delete an event (option == 3)

**STEP 6 (a)**: Prompt the user to enter the name of the event to delete

**STEP 6 (b)**: Read event name and verify its existence

**STEP 6 (c)**: Remove the event from the system

**STEP 6 (d)**: Display a success message

**STEP 7**: If the user chooses to register attendees for an event (option == 4)

**STEP 7 (a)**: Prompt the user to enter the name of the event

**STEP 7 (b)**: Read the event name and verify its existence

**STEP 7 (c)**: Prompt the user to enter the name of the attendee

**STEP 7 (d)**: Add the attendee to the event's attendee list

**STEP 7 (e)**: Display a success message

**STEP 8**: If the user chooses to view the event schedule (option == 5)

**STEP 8 (a)**: Prompt the user to enter the event name

**STEP 8 (b)**: Read the event name and verify its existence

**STEP 8 (c)**: Display the list of attendees

**STEP 8 (d)**: Prompt the user to select an action (update or delete an attendee)

**STEP 8 (e)**: If updating, prompt for the attendee name and new name, then update the list

**STEP 8 (f)**: If deleting, prompt for the attendee name and remove it from the list

**STEP 8 (g)**: Display a success message

**STEP 9**: If the user chooses to print the event schedule (option == 6)

**STEP 9 (a)**: Check if there are any events

**STEP 9 (b)**: Display event details: Name, Date, Time, Location, Description, Attendees

**STEP 10**: Repeat STEP 2 to STEP 9 until the user selects option 7

**STEP 11**: If the user chooses to exit (option == 7)

**STEP 11 (a)**: Display exit message

**STEP 11 (b)**: End program

**STEP 12**: End

Flowchart

**图示

AI 生成的内容可能不正确。General Flowchart**

**Specific Flowchart**

**add\_event**

图示

AI 生成的内容可能不正确。

**图示

AI 生成的内容可能不正确。update\_event**

**delete\_event**

图示

AI 生成的内容可能不正确。

**register\_attendee**

图示

AI 生成的内容可能不正确。

图示

AI 生成的内容可能不正确。**manage\_attendees**

**update\_attendee**

图示

AI 生成的内容可能不正确。

**delete\_attendee**

图示

AI 生成的内容可能不正确。

**print\_schedule**

图示

AI 生成的内容可能不正确。

Coding

from datetime import datetime

class EventHub:

def \_\_init\_\_(self):

self.events = {}

self.attendees = {}

with open("events.txt", "a") as file:

file.close()

self.read\_events()

def store\_events(self):

with open("events.txt", "w") as file:

for name, details in self.events.items():

attendees\_str = ",".join(self.attendees.get(name, []))

file.write(f"{name} | {details['date']} | {details['time']} | {details['location']} | {details['description']} | {attendees\_str}\n")

def read\_events(self):

with open("events.txt", "r") as file:

for line in file:

parts = line.strip().split(" | ")

name, date, time, location, description = parts[:5]

attendees = parts[5].split(",") if len(parts) > 5 else []

self.events[name] = {"date": date, "time": time, "location": location, "description": description}

self.attendees[name] = attendees

def conflict(self, date, time, location, exclude=None):

for event, details in self.events.items():

if event != exclude and details["date"] == date and details["time"] == time and details["location"] == location:

return True

return False

def add\_event(self):

while True:

name = input("Enter event name (Required, or type 'exit' to cancel): ").strip()

if name.lower() == "exit":

print("❌ Event creation canceled.")

return

if name:

break

print("Error: Event name cannot be empty. Please enter a valid name.")

while True:

date = input("Enter event date (YYYY-MM-DD, Required, or type 'exit' to cancel): ").strip()

if date.lower() == "exit":

print("❌ Event creation canceled.")

return

try:

datetime.strptime(date, "%Y-%m-%d")

break

except ValueError:

print("Error: Invalid date format. Please use 'YYYY-MM-DD'.")

while True:

time = input("Enter event time (e.g., 7:00AM-10:00PM, Required, or type 'exit' to cancel): ").strip()

if time.lower() == "exit":

print("❌ Event creation canceled.")

return

try:

start, end = time.split("-")

datetime.strptime(start.strip(), "%I:%M%p")

datetime.strptime(end.strip(), "%I:%M%p")

break

except (ValueError, IndexError):

print("Error: Invalid time format. Please use '7:00AM-10:00PM'.")

location = input("Enter event location (Required, or type 'exit' to cancel): ").strip()

if location.lower() == "exit":

print("❌ Event creation canceled.")

return

while not location:

print("Error: Event location cannot be empty. Please enter a valid location.")

location = input("Enter event location (Required): ").strip()

description = input("Enter event description (Optional, or type 'exit' to cancel): ").strip()

if description.lower() == "exit":

print("❌ Event creation canceled.")

return

if self.conflict(date, time, location):

print("Error: Time and location conflict with another event.")

return

self.events[name] = {

"date": date,

"time": time,

"location": location,

"description": description

}

self.attendees[name] = []

print(f"✔ Event '{name}' added successfully!")

self.store\_events()

def update\_event(self):

name = input("Enter event name to update: ").strip()

if name not in self.events:

print("Event not found.")

return

print("1. Name\n2. Date\n3. Time\n4. Location\n5. Description")

choice = input("Enter choice: ").strip()

fields = ["name", "date", "time", "location", "description"]

if choice == "1":

new\_name = input("Enter new event name: ").strip()

if new\_name in self.events:

print("Error: Event name already exists.")

return

self.events[new\_name] = self.events.pop(name)

self.attendees[new\_name] = self.attendees.pop(name)

else:

field = fields[int(choice) - 1]

new\_value = input(f"Enter new {field}: ").strip()

temp = self.events[name].copy()

temp[field] = new\_value

if self.conflict(temp["date"], temp["time"], temp["location"], exclude=name):

print("Error: Time and location conflict with another event.")

return

self.events[name][field] = new\_value

print("✔ Event updated successfully!")

self.store\_events()

def delete\_event(self):

if not self.events:

print("No events available to delete.")

return

while True:

name = input("Enter the event name to delete (or 'exit' to cancel): ").strip()

if name.lower() == "exit":

print("❌ Event deletion canceled.")

return

if name in self.events:

del self.events[name]

del self.attendees[name]

print(f"✔ Event '{name}' deleted successfully!")

self.store\_events()

return

print("Error: Event not found. Try again.")

def register\_attendee(self):

if not self.events:

print("No events available to register for.")

return

while True:

name = input("Enter the event name to register for (or 'exit' to cancel): ").strip()

if name.lower() == "exit":

print("❌ Registration canceled.")

return

if name in self.events:

break

print("Error: Event not found. Try again.")

attendee\_names = input("Enter attendee name (comma-separated): ").strip()

if attendee\_names:

attendees = [attendee.strip() for attendee in attendee\_names.split(",") if attendee.strip()]

if attendees:

self.attendees[name].extend(attendees)

print(f"{', '.join(attendees)} registered successfully for '{name}'!")

self.store\_events()

return

print("Error: Attendee names cannot be empty. Try again.")

def manage\_attendees(self):

name = input("Enter the event name to manage attendees: ").strip()

if name not in self.attendees:

print("Error: Event not found.")

return

print("1. Update Attendee Name")

print("2. Delete Attendee")

choice = input("Enter your choice (1-2): ").strip()

if choice == "1":

self.update\_attendee(name)

elif choice == "2":

self.delete\_attendee(name)

else:

print("Error: Invalid choice.")

def update\_attendee(self, event\_name):

print("Current Attendees:", ", ".join(self.attendees[event\_name]) if self.attendees[event\_name] else "None")

old\_name = input("Enter the attendee name to update: ").strip()

if old\_name not in self.attendees[event\_name]:

print("Error: Attendee not found.")

return

new\_name = input("Enter the new name: ").strip()

if new\_name:

idx = self.attendees[event\_name].index(old\_name)

self.attendees[event\_name][idx] = new\_name

print(f"✔ Attendee '{old\_name}' updated to '{new\_name}'!")

self.store\_events()

else:

print("Error: New name cannot be empty.")

def delete\_attendee(self, event\_name):

print("Current Attendees:", ", ".join(self.attendees[event\_name]) if self.attendees[event\_name] else "None")

name = input("Enter the attendee name to delete: ").strip()

if name in self.attendees[event\_name]:

self.attendees[event\_name].remove(name)

print(f"✔ Attendee '{name}' removed!")

self.store\_events()

else:

print("Error: Attendee not found.")

def print\_schedule(self):

if not self.events:

print("No events available.")

return

print("\n Event Schedule:")

for name, details in self.events.items():

print("=" \* 40)

print(f"Name: {name}")

print(f"Date: {details['date']}")

print(f"Time: {details['time']}")

print(f"Location: {details['location']}")

print(f"Description: {details.get('description', 'N/A')}")

attendees = self.attendees.get(name, [])

if attendees:

print("Attendees:")

for attendee in attendees:

print(f" - {attendee}")

else:

print("Attendees: None")

print("=" \* 40)

def main\_menu(self):

while True:

print("\n" + "=" \* 50)

print("✨EVENTHUB - EVENT MANAGEMENT SYSTEM✨ ".center(50))

print("=" \* 50)

print(" 1 - Add Event")

print(" 2 - Update Event")

print(" 3 - Delete Event")

print(" 4 - Register Attendee")

print(" 5 - Manage Attendees")

print(" 6 - Print Event Schedule")

print(" 7 - Exit")

print("-" \* 50)

choice = input("Enter your choice: ").strip()

print("-" \* 50)

if choice == "1":

self.add\_event()

elif choice == "2":

self.update\_event()

elif choice == "3":

self.delete\_event()

elif choice == "4":

self.register\_attendee()

elif choice == "5":

self.manage\_attendees()

elif choice == "6":

self.print\_schedule()

elif choice == "7":

print("\n" + "\_" \* 50)

print(" Exiting EventHub... Goodbye! ".center(50))

print("\_" \* 50 + "\n")

input("\nPress Enter to exit...")

break

else:

print("Error: Invalid choice. Try again.")

hub = EventHub()

hub.main\_menu()

Test Data

|  |  |  |
| --- | --- | --- |
| **No.** | **Test Case** | **Output** |
| **Main Menu Tests** | | |
| 1 | Input option from menu (valid)  Enter  1 |  |
| 2 | Input option from menu (valid)  Enter  2 |  |
| 3 | Input option from menu (valid)  Enter  3 |  |
| 4 | Input option from menu (valid)  Enter  4 |  |
| 5 | Input option from menu (valid)  Enter  5 |  |
| 6 | Input option from menu (valid)  Enter  6 |  |
|  | Input option from menu (valid)  Enter  7 |  |
| 7 | Input option from menu (invalid)  Enter  9  abc |  |
| **Event Creation Tests** | | |
| 8 | Create event with valid details  Enter  Name: music festival  Date: 2025-04-01  Time:10:00AM- 12:00PM  Location: Central Plaza  Description: UNMUTE: Battle of the Bands |  |
| 9 | Create event with empty name  Enter  Name: (blank) |  |
| 10 | Create event with invalid date  Enter  2025abc  2025-01-32  29-10-2025  (empty input) |  |
| 11 | Create event with invalid time format  Enter  13:00-14:00  9:70AM-10:70AM  15:00AM-10:00PM  (empty input) |  |
| 12 | Create event with empty location  (empty input) |  |
| 13 | Attempt to add a conflicting event  Enter  Foundation Ball  2025-05-01  9:00AM-10:00PM  UG Room |  |
| **Event Update Tests** | | |
| 14 | Update event name with valid value  Enter  2  Music Festival  1  34th Music Festival |  |
| 15 | Update event date with valid value  Enter  2025-04-07 |  |
| 16 | Update event time with valid value  Enter  3:00PM-6:00PM |  |
| 17 | Update event location with valid value  Enter  Club |  |
| 18 | Update event description with valid value  Enter  New |  |
| 19 | Update event with empty name  (empty input) |  |
| 20 | Update non-existent event  Enter  Meeting1 |  |
| **Event Deletion Tests** | | |
| 21 | Delete existing event  Enter  CAN Workshop |  |
| 22 | Delete non-existent event  Enter  OSM Meeting |  |
| 23 | Attempt to delete without selecting  (empty input) |  |
| **Attendee Registration Tests** | | |
| 24 | Register multiple attendees  Enter  Isha, Xinyi, Emily |  |
| 25 | Register with empty name  (empty input) |  |
| **Attendee Management Tests** | | |
| 26 | Update attendee name  Enter  Music Festival  1  Xinyi  Liu Xinyi |  |
| 27 | Attempt to update an attendee name in a non-existent event  Enter  OSM |  |
| 28 | Remove attendee name  Enter  34th Music Festival  2  Maya |  |
| 29 | Remove attendee name with empty input  Enter  34th Music Festival  (empty input) |  |
| 30 | Attempt to update a non-existent attendee name  Enter  c |  |
| 31 | Attempt to delete a non-existent attendee name  Enter: Amber |  |

Output

**Valid Data**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**Invalid Data**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**

**文本

AI 生成的内容可能不正确。**