



# CW Passport Registry Database

ANGELA XU

# INTRODUCTION

## Design Overview

CW Passport Registry is a resourceful database program used to record and register students for Claude Watson passport events. Users, such as the Claude Watson ambassadors, can add and delete events as well as add, delete, and modify participant information in the events using simple commands.

The base class, Event, contains a vector of participants, acting as a list. Within the class, there are more functions that modify and add Participant information into the vector: listInfo, addInfo, deleteInfo, modifyInfo. There are also three subclasses, Online, Hall, and Room, each representing events hosted in locations online, in Cringan Hall, or in a classroom. Each subclass contains a specific registration limit in relation to the actual size of the locations (Maximum capacity of Online events - 100, Hall events - 300, Room events - 50).

The user is prompted to enter a single digit integer to choose their action at the beginning. There are also many user-friendly titles and headers for each function the user decides to perform. The user may then follow the on-screen instructions and information to make their inputs. All inputs are validated immediately and exceptions are thrown using functions from the Exception class, which inherits from the exceptions class the c++ library provides. Afterwards, exceptions are caught and displayed using the displayMessage function of the Exception class.

After exiting the program, all used memory and vectors are freed to prevent memory leakage. There are destructors in both the Event and Participant class, which clears vectors and other information.

## Traceability Matrix

[illegible]

# SYSTEM ARCHITECTURE DESIGN

## Program Architecture

### **Main Program:**

- Contains vector<vector<Event>\*> which is a vector that holds pointers to Event objects
- Contains other variables such as:
  - Three vector<Event>, each representing vectors that hold Online Events, Hall Events, and Room events
  - String variables that hold event name, event department, participant name, participant department, participant email, event location name, and the user input command
  - Integer variables that hold participant's number of stamps, participant grade, and some indexes for future use (will be described later on)
  - Boolean variable "flag", which controls the number of iterations of the while loop and stops the program when the user wants to exit
- Program uses a while loop to iterate until user stops iteration via the exit command
- Using a switch statement, proceed to code according to the command the user inputs to perform their chosen function:
- If the user selects **"Add Event"**, this case is located using the "ADDEV" enumerator. A new object of Online/Hall/Room type subclass will be made and stored in the correct vector array
  - onlineev: holds pointers to participant objects who will be attending online meetings. Initial capacity is 0 and maximum capacity is 100.
  - hallev: holds pointers to participant objects who will be attending an event at Cringan Hall. Initial capacity is 0 and maximum capacity is 300.
  - classev: holds pointers to participant objects who will be attending an event in the drama classroom or a similar-sized classroom. Initial capacity is 0 and maximum capacity is 50

## Main Program Continued:

- If the user selects **"Add Participant"**, this case is located using the "ADDPAR" enumerator.
  - An existing Participant object is modified according to the user's input about the participant name, department, number of stamps, grade, and email.
  - The copied information of that object is put into a specific Event vector as indicated by the user.
  - Before pushing the object into the vector, the input is validated by checking if there is already a Participant with the same name in the vector.
  - Verifies whether the Participant is eligible for a Claude Watson stamp: if the Participant's department and the Event's department are not the same, and if the Participant is in Claude Watson, then the Participant's number of stamps increases by 1, with restrictions that cannot overstep the maximum of 8 stamps.
- If the user selects **"Modify Participant"**, this case is located using the "MODPAR" enumerator.
  - Using functions from the Event class and other methods, the indexes of the input Event name are found.
  - A Menu is displayed to ask the Participant what information they want to modify.
  - If the user chooses to modify the name, stamps, grade, or email, an existing Participant object is modified to only change the input information.
  - If the user chooses to modify the Event attending, find the current event name and the new event name and their respective index locations. Delete the current participant in the event and add a new participant in the new event with the same participant information. Exception will be thrown if there is already a Participant with the same name attending the new event.
  - If the user does not enter any options from the menu, an exception is thrown.

## Main Program Continued:

- If the user selects **"Delete Participant"**, this case is located using the "DELPAR" enumerator.
  - The location of the Event the Participant is attending is found using the Event name, and the Participant is deleted from the event.
- If the user selects **"Delete Event"**, this case is located using the "DELEV" enumerator.
  - The location of the Event is found using the input Event name, and the Event is deleted from the event vector.
- If the user selects **"Exit"**, this case is located using the "EXIT" enumerator.
  - The exit message is displayed and the while loop is stopped using a boolean flag variable.
- If the user makes a mistake in the formatting or syntax during the process, an exception is thrown.

## Exception Class:

- Inherits from the public exception c++ library
- Constructor initializes Exception class attribute: message
- Contains a string msg variable that stores an error message
- getMessage function returns the error message
- displayMessage function displays the Exception Error using user-friendly interface
- Destructor frees memory at the end of the program's execution

## Super Class Event:

- Contains properties that can be inherited by subclasses
- Constructor initializes all event object attributes (i.e. name of event, department type hosting the event) and stores a vector of Participant objects
- Destructor frees memory at the end of the program's execution through clearing the participant vector
- Contains all functions available to the user:
  - addInfo: assigns object's attributes via validated user input
  - listInfo: prints all object attributes in a user-friendly manner
  - modifyInfo: allows user to make changes to an existing object within the participant vector
  - deleteInfo: Allows the user to delete a participant object, its pointer and information. Also deletes the object from the appropriate vector array. First checks whether a Participant's name exists in the participant vector, otherwise throws an Exception object if it is not found.
- Contains functions that are used in the main class to find certain information:
  - getName: returns Event name
  - getDepartment: returns Event department
  - isEmpty: returns true if the participant vector of the Event is not empty. False if the participant vector is empty
  - findParpos: returns the participant position in the vector. Returns -1 if the participant's name is not found
  - getParName: returns a Participant's Name using the index of the vector given and using functions part of the Participant class
  - getParDep: returns Participant Department (i.e. Music, Drama, Film, Visual Art, Dance)
  - getParEmail: returns the Participant email address using the index of the vector given and using functions part of the Participant class
  - getParStamp: returns the number of Claude Watson stamps a Participant has
  - getParGrade: returns the Participant Grade (i.e. 9, 10, 11, 12)

## Subclasses:

- Each subclass is an event location type within the event class :
  - Online (ex. Zoom)
  - Hall (ex. Cringan Hall)
  - Room (ex. Drama Room)
- Each subclass contains a constructor to initialize their respective objects
- The Destructor (and other functions) are inherited from the destructor in the superclass Event. It frees up memory by clearing the participant vector once an object of their type is deleted
- Other virtual functions that are inherited from the Superclass Event are addInfo and listInfo
- Each subclass contains different properties and restrictions on the number of people who can sign up for the event.
- Limitations specific to each subclass: Zoom meetings can only contain 100 people, Cringan Hall can contain up to 300 people through estimates, and the Drama Room can contain 50 guests.

## Class Participant:

- Constructor initializes and stores all participant attribute variables (i.e. name, number of Claude Watson stamps, department of participant, email, grade)
- Destructor frees memory at the end of the program's execution
- Contains functions that are used in the program to get participant information:
  - listParInfo: displays all participant information
  - modInfo: modifies participant information in respect to the given information
  - getName: returns Participant name
  - getDep: returns Participant department
  - getEmail: returns Participant's email address
  - getStamp: returns the number of stamps the Participant owns
  - getGrade: returns Participant grade



## Other Functions:

- choicesMenu: prints the initial Program selection menu for users
- welcomeMessage: print home Welcome message
- modparMenu: prints the selection menu for the Modify Participant selection
- Title statements:
  - addevTitle: prints the title for the Add Event selection
  - addparTitle: prints the title for the Add Participant selection
  - modparTitle: prints the title for the Modify Participant selection
  - displayinfoTitle: prints the title for the Display Information selection
  - delparTitle: prints the title for the Delete Participant selection
  - deletTitle: prints the title for the Delete Event selection
- isEmpty: returns whether a vector<vector<Event>\*> has no Participants registered in any events (uses the function isEmpty from the Event superclass)
- noEvent: returns whether a vector<vector<Event>\*> has no Events in each of the vectors
- displayInfo: prints all the events within each of the subcategories Online, Hall, and Room. Also prints all the participants within each of the events.
- findEventloc: returns the event location of a specific event. Throws an exception if the event name is not in the database.
- findEventindex: returns the event index within a vector of an event location. Throws an exception if the event name is not in the database.
- emailCheck: validates the email address
- gradeCheck: validates the grade
- stampCheck: validates the number of stamps (maximum is 8)

## Layered Pattern

Program can be broken down into subtasks:

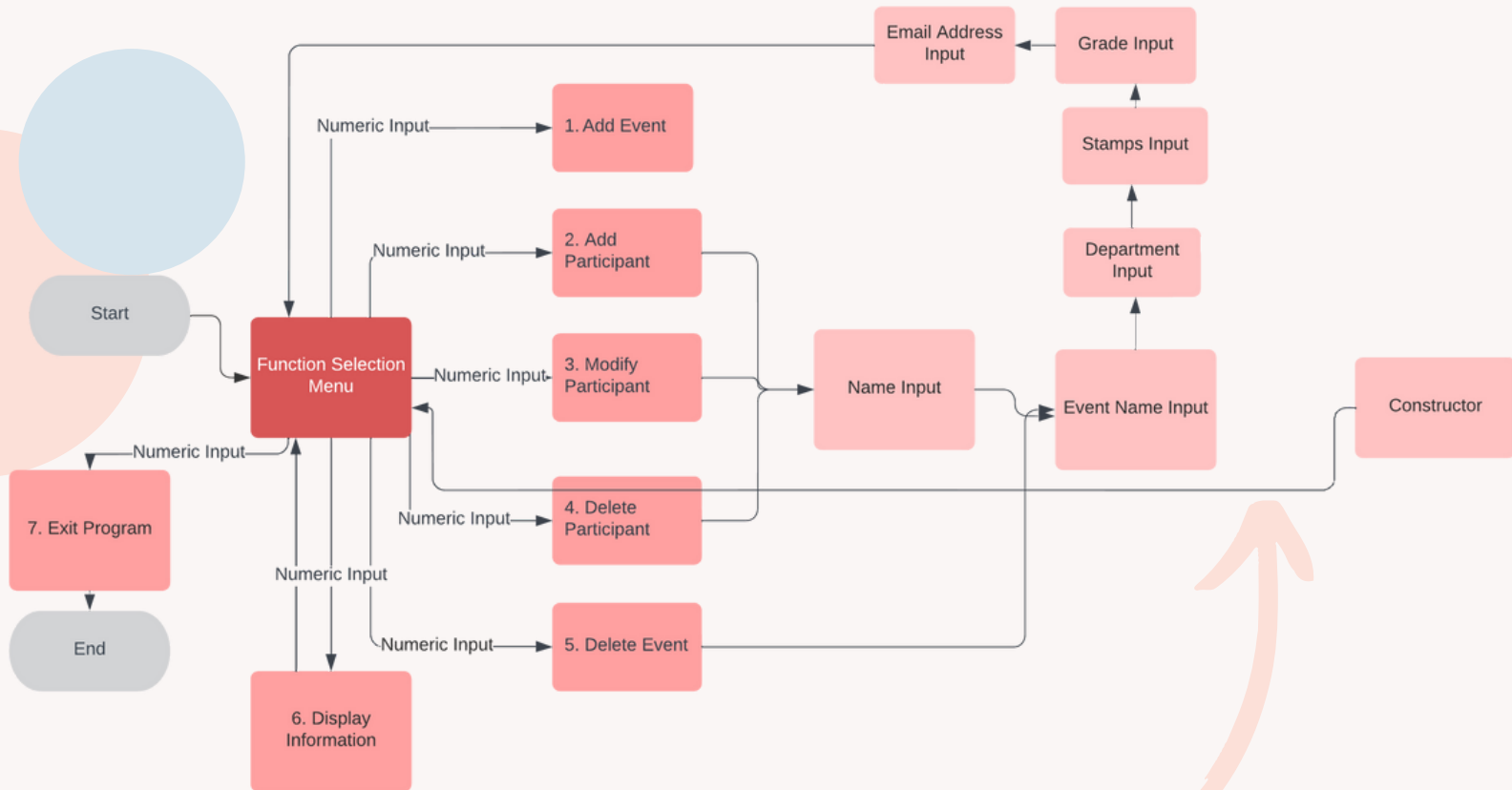
- **Layer 1** is the Presentation Layer:
  - Display welcoming message
  - Display selection menu in a while loop
  - Allows user to create objects, assigning them to a pointer and a respective vector array
  - Ensure all fields have been initialized for execution in Layer 2
- **Layer 2** is the Business Layer:
  - Listing events and the attributes of the people attending the events
- **Layer 3** is Database Layer:
  - Storing event and participant information in vector arrays
  - Modifying or deleting objects will occur in this layer

### **Alternative Designs**

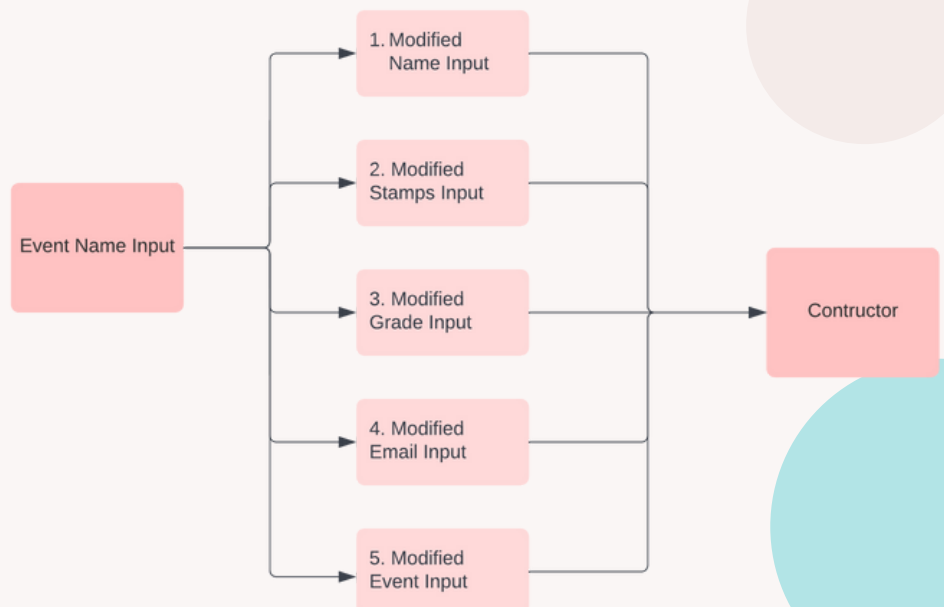
An alternative design is using the Master-Slave pattern, where the Master database stores all the information and reading and displaying the code is used by the slave. This architecture is good because the Master database will be reliable and will stabilize a system. Some disadvantages of Master-Slave are its need for handling data transfer and synchronization across the master and slave databases. Since I would need to create event classes and participant classes, replicating data across the databases would be tedious and not as efficient as using a layered structure.

Another alternative design would be the model-view-controller pattern. The model will contain all the core data and objects, the view component will be displaying welcome messages and the menu, and the controller handles the input from the user. Some advantages of this architecture are its easily modifiable structures and it can be easily debugged due to its layers. However, the MVC pattern is too complex to be used for small programs, and would be better used for organizing large web applications.

# Systems Interface Design



Due to not having enough space, the Systems Interface design in-between Event Name Input and Constructor are the following:



# COMPONENTS

## Classes:

- Base class Event contains functions that are used for user selection
  - Subclasses
    - Subclass Online used to initialize and destruct Event objects with event type Online
      - Inherits attributes for objects from functions in base class Event
    - Subclass Hall used to initialize and destruct Event objects with event type Hall
      - Inherits attributes for objects from functions in base class Event
    - Subclass Room used to initialize and destruct Event objects with event type Room
      - Inherits attributes for objects from functions in base class Event

## Structures:

- Vector "events" holds pointers to vectors of objects from superclass Event types
- Vector onlineev holds objects from subclass event type Online
- Vector hallev holds objects from subclass event type Hall
- Vector roomev holds objects from subclass event type Room
- All vectors start at capacity 0 and grow dynamically by 1 each time an object is added into the vector

## Constructor:

- Each class contains a constructor. Each constructor is used when initializing a new object of the Participant class or Event class.
  - Subclass constructors receive input and inherit from the Superclass constructor

## Destructor:

- All class destructors are invoked when the user chooses to exit the program.
- Vector arrays are cleared of their pointers and objects to free allocation
  - Clearing the objects and the pointers of objects within vector arrays calls upon their respective destructors

## Pointers:

- Vector “events” take in pointers of `vector<Event>*` type in order to access the direct location of each subclass vector
- All pointers are deleted at the end of the program

## Functions:

- Event Class Methods:
  - `listInfo`, `getName`, `getDepartment`, `isnotEmpty`, `addInfo`, `deleteInfo`, `modifyInfo`, `findParpos`, `getParName`, `getParDep`, `getParEmail`, `getParStamp`, `getParGrade`
- Event Subclass Methods:
  - `addInfo`, `listInfo`
- Participant Class Methods:
  - `listParInfo`, `modInfo`, `getName`, `getDep`, `getEmail`, `getStamp`, `getGrade`
- Other functions:
  - `choicesMenu`, `welcomeMessage`, `modparMenu`, `adddevTitle`, `addparTitle`, `modparTitle`, `displayinfoTitle`, `delparTitle`, `delevTitle`, `isEmpty`, `noEvent`, `displayInfo`, `findEventloc`, `findEventindex`, `emailCheck`, `gradeCheck`, `stampCheck`

## Statements:

- Switch statement:
  - `int Main` has a switch statement printing the `mainMenu` the user can choose 7 options from: Add Event, Add Participant, Modify Participant, Delete Participant, Delete Event, Display Information, and Exit Program

- If statements:
  - int Main has an if statement used for the Modify Participant function. A Menu is printed and the user can choose from 5 options to modify Participant information with: Name, Number of Stamps, Grade, Email, Event Attending.
  - If the user chooses the first 4 options, the program asks for the new information, which is then modified.
  - If the user chooses the fifth option, the program asks for the new Event name and searches for the new Event location. The participant is deleted and then added into the new Event.
  - Used for input validation in many functions. Throws a specific Exception when the input is invalid.
  - Used for checking if the Participant is being added or moved to an Event currently containing a Participant with the same name. A specific Exception will be thrown if so.

## Loops:

- While loop used to iterate code numerous times
  - Main while loops over program selection until user invokes the exit function
- For loops used to iterate through each component of a variable
  - Used to iterate through all participants in an event vector
  - Used to iterate through all events of an event vector
- Try and catch blocks
  - Try and catch block used at the end of the while loop to catch any exceptions thrown while the program is running

# Variables

## **int Main:**

- bool flag: used to control the number of iterations in the while loop. This variable changes to false when the user chooses to exit the program
- vector<vector<Event>\*> events: used to gather all event and event vectors in one variable
- vector<Event> onlineev: only Online type objects are pushed into this vector. The pointer of this vector is in the 0 index of events
- vector<Event> hallev: only Hall type objects are pushed into this vector. The pointer of this vector is in the 1 index of events
- vector<Event> roomev: only Room type objects are pushed into this vector. The pointer of this vector is in the 2 index of events
- String evname: used for storing Event Name
- String evdep: used for storing Event Department
- String parname: used for storing Participant Name
- String pardep: used for storing Participant Department
- int parstamps: used for storing the Participant's number of stamps
- int pargrade: used for storing the Participant's grade
- String paremail: used for storing the Participant's email address
- int loc: used for storing the index of a vector<Event>\* in events
- String locname: used for identifying the event location of an event within the events
- int index: used for storing the index of an Event in vector<Event>
- int locnew: used for storing the new loc for the Modifying Participant selection
- int indexnew: used for storing the new index for the Modifying Participant selection
- int indexPar: used for storing the index of the participant within an Event object's participants vector
- Participant p: initialized with the parameters Name (""), Department (""), Number of stamps (0), Grade (0), Email (""). Used for adding a new Participant or modifying a participant by modifying its information.
- String command: used for getting and storing user input



## Event class:


- String name: used to store Event name
- String department: used to store Event department
- vector<Participant> participants: used to store all the participants registered in an Event



## Event Subclass:

- Int maxParticipants: used to store each number of maximum registrants of an Event subclass type

## Participant class:

- String name: used to store Participant name
  - String department: used to store Participant department
  - int stamps: used to store number of Participant stamps
  - int grade: used to store Participant's grade
  - String email: used to store Participant's email address
- 



# USER INTERFACE DESIGN

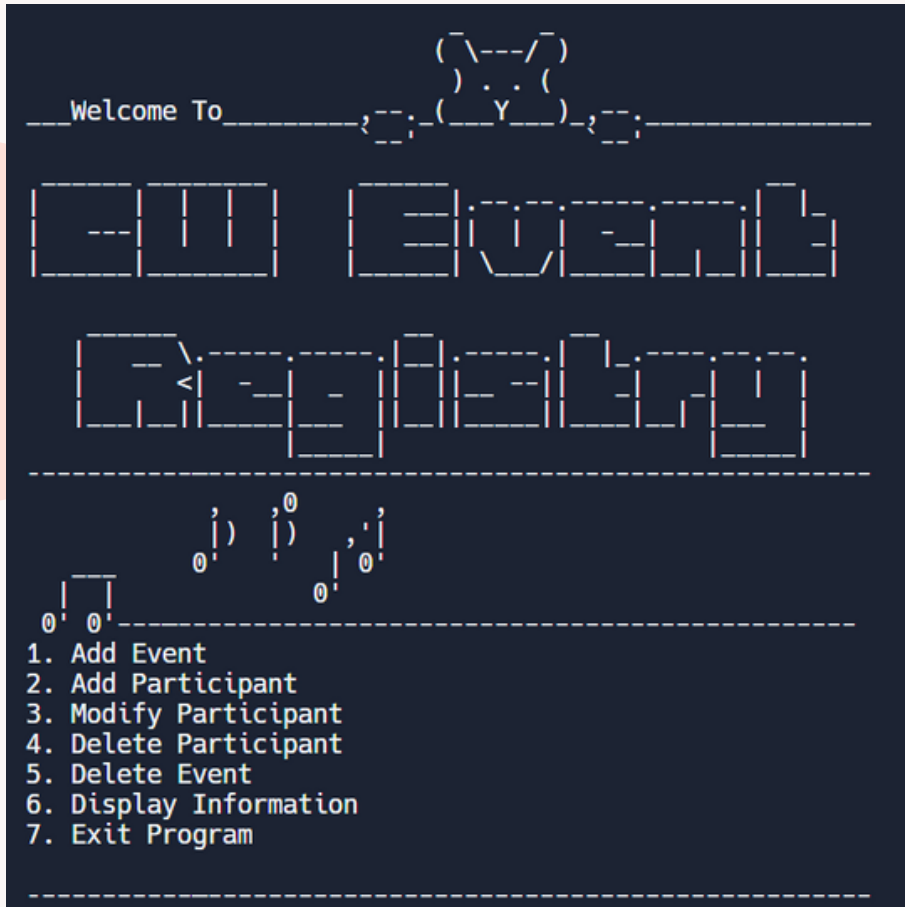


Figure 1: Logo and Menu Output with a teddy bear and music notes, representing the Claude Watson community.

```
catch(Exception &e){
    e.displayMessage();
}
catch(exception &e){
    Exception obj(e.what());
    obj.displayMessage();
}
```

Figure 2: Program's code to how Exceptions are caught and displayed

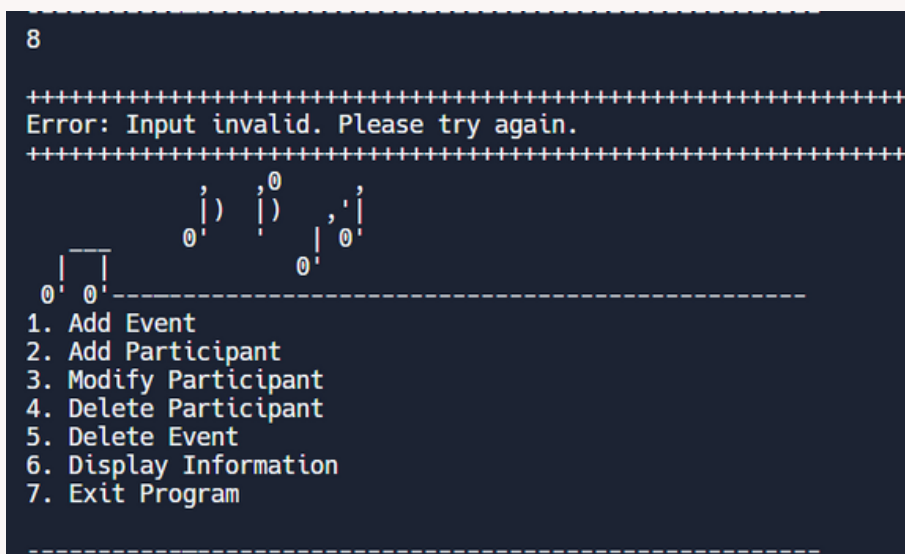


Figure 3: Program Selection Out of Range. Error messages are surrounded by "+" for easy readability.

All Errors are detailed and tell the user exactly why their input received an error. Otherwise, errors are still displayed due to the base case of the catch function.

After the error is caught, the current selection is terminated and the program selection menu will be re-displayed.



```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): Music
Number of CW event stamps: 0
Grade: 12
Email: angela.xu@gmail

+++++
Error: Email invalid. There must be a '.' symbol after the '@'.
+++++

```

Figure 8: Possible error when creating a Participant; invalid email

Figure 9: Possible error when creating a participant; invalid email part 2

```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): Music
Number of CW event stamps: 0
Grade: 12
Email: angelagmail

+++++
Error: Email invalid. Email must contain '@' and '.' symbols.
+++++

```

```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): No

+++++
Error: Please enter a valid department type. (Case sensitive)
+++++

```

Figure 10: Possible error when creating a participant; invalid department

```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): Music
Number of CW event stamps: 9
Congratulations! You have exceeded the maximum number of stamps (8) and cannot receive more!
Grade: 2

+++++
Error: Grade invalid. Only students can register in events. Please enter a grade from 9 to 12.
+++++

```

Figure 11: Possible error when creating a participant; invalid grade, and exceeding maximum stamp notification

```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): Music
Number of CW event stamps: -1

+++++
Error: Stamp number invalid. Please enter a number from 0 to 8. (Each Claude Watson student must graduate with 8 stamps;
2 stamps per year)
+++++

```

Figure 12: Possible error when creating a participant; invalid number of stamps

```

=====
                        DISPLAY INFORMATION
=====

ONLINE EVENTS:
  Registrants in Event Music Gala:
  No Participants in this event.

HALL EVENTS:
  Registrants in Event Film Festival:
  No Participants in this event.

ROOM EVENTS:
  Registrants in Event One Acts:
  No Participants in this event.

```

Figure 13: Display Information of 3 events with no participants yet. Each Event is indented for clarity and separated into categories for Online Events, Hall Events, and Room Events

```

=====
                        ADD PARTICIPANT
=====
Name: Angela Xu
Department (Music, Visual, Drama, Film, Dance, N/A): Music
Number of CW event stamps: 2
Grade: 12
Email: angela@gmail.com
Which Event are you attending? (Name): Film Festival
***Event Department and Participant Department not the same. Participant is eligible for another Claude Watson stamp!***
Registered Participant: ANGELA XU to Event: Film Festival!

```

Figure 14: A successful registration of a participant into the event "Film Festival"

Figure 15: This is the user interface when a user selects the Display Information function. Registrant names are capitalized and information is displayed clearly. Note participant "Angela Xu" has 1 more stamp than when initialized because Participant was eligible for another stamp in Figure 14.

```

ONLINE EVENTS:
  Registrants in Event Music Gala:
  No Participants in this event.

HALL EVENTS:
  Registrants in Event Film Festival:
  -----
Participant Name: ANGELA XU
Department of Major: Music
Number of Stamps: 3
Grade: 12
Email: angela@gmail.com
  -----

ROOM EVENTS:
  Registrants in Event One Acts:
  No Participants in this event.

```



Figure 16: Note occur at the same time with a different

ONLINE EVENTS:  
Registrants in Event Music Gala:  
No Participants in this event.

HALL EVENTS:  
Registrants in Event Film Festival:

---

Participant Name: ANGELA XU  
Department of Major: Music  
Number of Stamps: 3  
Grade: 12  
Email: angela@gmail.com

---

Registrants in Event Gallery:

---

Participant Name: COOL KID  
Department of Major: N/A  
Number of Stamps: 0  
Grade: 12  
Email: colkid@gmail.com

---

ROOM EVENTS:  
Registrants in Event One Acts:  
No Participants in this event.

```
-----
3
=====

                MODIFY PARTICIPANT

=====

What is the current Participant Name?: Cool Kid
What is the Event Name the Participant is currently attending?: Gallery
\[]
||
||
||
||
||
||
\::')
/::\
\__:/ -----
What would you like to modify in Participant? (Department cannot be changed due to stamp calculations.)
1. Name
2. Number of Stamps
3. Grade
4. Email
5. Event Attending

-----
4
Current Email: colkid@gmail.com
New Email: coolkid@gmail.com
Successfully modified Participant Information!
```

```

\::')
/::\
\_:/ -----
What would you like to modify in Participant? (Department cannot be changed due to stamp calculations.)
1. Name
2. Number of Stamps
3. Grade
4. Email
5. Event Attending

-----
4
Current Email: colkid@gmail.com
New Email: coolkid@gmail.com
Successfully modified Participant Information!

```

The Modification menu is displayed and the user can select the modification they would like to make. The current information is displayed for clarity. After the user information is updated and all input is validated, the program displays a success message to indicate the end of the modification process.

```

2
=====
                                ADD PARTICIPANT
=====
Name: Coolest Kid
Department (Music, Visual, Drama, Film, Dance, N/A): N/A
Number of CW event stamps: 0
Grade: 12
Email: coolest@gmail.com
Which Event are you attending? (Name): cool stuffs
+++++
Error: Event does not exist in the database.
+++++

```

Figure 18: If an event is not located in any of the vector arrays, an error message is displayed. This error message is also applicable to the Modify Participant, Delete Participant, and Delete Event functions.

Figure 19: A successful deletion of a Participant

```

=====
                                DELETE PARTICIPANT
=====
What is the current Participant Name?: Cool Kid
What is the Event Name the Participant is currently attending?: Gala
Successfully deleted Participant Cool Kid from Gala!

```

```

5
-----
                                DELETE EVENT
-----
What is the Event Name?: Gala
Successfully deleted Event Gala from Online Events!

```

Figure 20: A successful deletion of an Event

Figure 21: Possible Error when adding a Participant; Non- Claude Watson students cannot receive stamps

```

2
=====
                                ADD PARTICIPANT
=====
Name: COol stuffs
Department (Music, Visual, Drama, Film, Dance, N/A): N/A
Number of CW event stamps: 3
+++++
Error: Students not within the Claude Watson program cannot receive stamps.
+++++

```

```
-----
2
=====

ADD PARTICIPANT

=====

+++++
Error: Cannot create a Participant. Please add an Event first.
+++++
```

Figure 22, 23, 24, 25:  
Errors when no  
Participant or Event have  
been added yet. Cannot  
proceed with the next  
steps.

```
-----
3
=====

MODIFY PARTICIPANT

=====

+++++
Error: Cannot Modify Participant. Must have at least 1 Event and 1 Participant.
+++++
```

```
-----
4
=====

DELETE PARTICIPANT

=====

+++++
Error: Cannot Delete Participant. Must have at least 1 Event and 1 Participant.
+++++
```

```
-----
5
=====

DELETE EVENT

=====

+++++
Error: Cannot Delete. Must have at least 1 Event to delete.
+++++
```



Figure 26: Once the user exits, they receive a friendly goodbye message written with a font containing flowers.





## Testing

This program has been tested over 100 times from execution to finish. Testers include myself (Angela Xu), peers and other Claude Watson staff and students. After every test, the code was re-evaluated for efficiency and bugs, and modified accordingly. This code has been shortened as much as possible through repeated use of functions and inheritance of classes.

During initial tests, the user interface was not intuitive enough for users, hence, headers, titles, and section separators were added to make readability easier.

## Thank you!



I would like to extend my thanks to my mom for encouraging me and giving me moral support whenever bugs were discovered or additions needed to be made.

Thank you to Evelyn Phan and Catherine Tang for help with debugging, testing, and helping me through each coding session together.

Thank you to every tester and Mr. Noukhovitch for teaching me what kinds of projects and ideas can be achieved with code.

Finally, thank you for reading my report. Completing this project has been a great milestone in my life so displaying my work here and having people read it means a lot to me. Please look forward to my future projects as well. Thank you!



[www.linkedin.com/in/angela-xu](https://www.linkedin.com/in/angela-xu)



[angela.xu.dev@gmail.com](mailto:angela.xu.dev@gmail.com)







