GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP010-4-2-PWP

PROGRAMMING WITH PYTHON

UCDF2104BIT / UCDF2104ICT(SE) / UCDF2104ICT(DI) / UCDF2104ICT(ITR) / UCDF2104ICT

HAND OUT DATE: 13TH APRIL 2022

HAND IN DATE: 31st MAY 2022

WEIGHTAGE: 100%

INSTRUCTIONS TO CANDIDATES:

1. Submit your assignment online in MS Teams unless advised otherwise

2. Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld

3. Cases of plagiarism will be penalized

4. You must obtain at least 50% in each component to pass this module

ASIAN EVENT GENERAL INTERFACE SOLUTION

designed for Asian Event Management Service

AAPP010-4-2-PWP | UCDF2104ICT(SE)

13th April 2022 – 31st May 2022

> Prepared by:
AMADEA LIM YI WEN | TP064038
GAN MING LIANG | TP063338

> Supervised by:
AMARDEEP SINGH A/L UTTAM SINGH

Table of Contents

Introduction & Assumptions

> Introduction

Asian Event Management Service (AEMS) is an agency that assist clients in the organization and participation of events. Recently, AEMS have experienced a healthy growth in the use of their services and is looking to expand with a new interface in order to accommodate for the new surge in demands. As such, our team have set out and developed a program called **Asian Event General Interface Solution** (or **AEGIS)** to help ameliorate their services and enhance business processes.

AEGIS

AEGIS is an **online event management system** which aims to improve user experience through a menu-driven interface that is both simple and elegant. AEMS staffs can register as **Admins** to create events, modify events, and display record of all registered customers and associating payments; while customers can either proceed as **Guest** to view events details, or register as **Members** to add them into a cart and make payments to secure their spot in the corresponding event(s).

AEGIS is coded exclusively using **python** (version 3.10.2) and contains a total of 20 unique functions that can be summoned by inputting the respective numbers/letters into the terminal during execution of the program. AEGIS also comes with 4 text files known as “adminList.txt”, “memberList.txt”, “eventList.txt” and “cartList.txt” which will be used as a database to store user information. AEGIS is also fully customizable, making any further program amendments an ease of mind.

> Assumptions

AEGIS is developed under the assumptions that most clients at Asian Event Management Service would not be technically proficient enough to understand the jargons and programming terms used in python. As such, pseudocodes are created which features simple, short, English alternative to enlighten them in understanding the purpose of our codes. Flowcharts are also created for each of the 20 functions in AEGIS, which offers a great graphical representation of the process flow that's taking place during execution of AEGIS.

The menu-driven interface is also designed with the assumption that the users would be literate in English as all messages, instructions, functions, and features will be displayed solely in English. However, due to the simplistic and straight-forward nature of MDI, we do not expect our users to struggle when prompted with choices.

The last assumption is that the **try and except statements** can be utilized in the creation of AEGIS. Many other functions such as dict(), max(), min(), and sum() are deemed as inappropriate for use in this assignment, which has challenged us to find creative alternative to overcome this problem. However, the try and except function are an essential feature in AEGIS as it would prevent the program from crashing whenever a ValueError is detected. As such, it is irreplaceable with any other form of codes, so we hope that the use of the try and except statement is not illicit and in complete accordance with the assessment requirements.

Design of the Program

> Homepage design

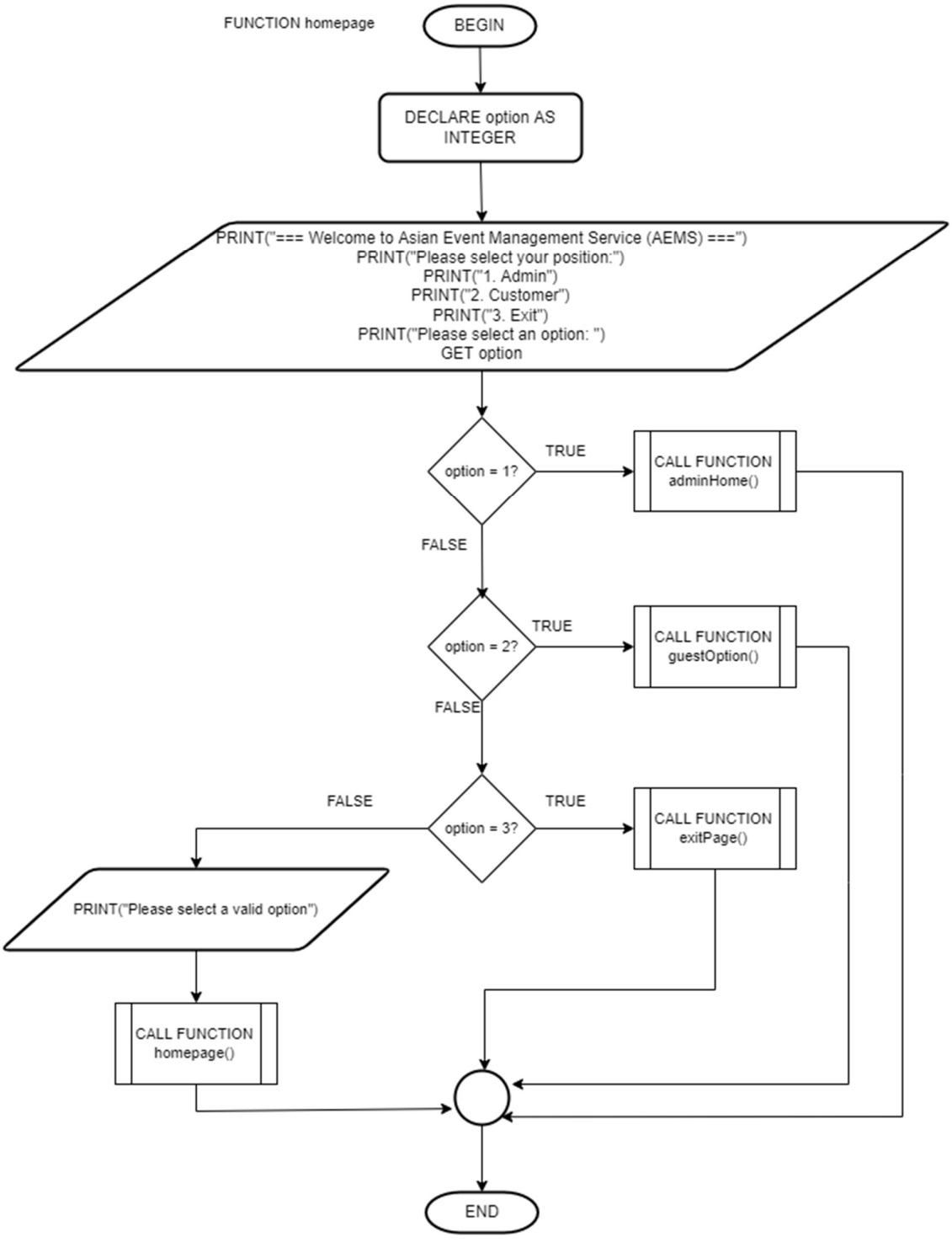
Homepage is the page that users will first see when they launch the AEGIS program. It serves the purpose of defining the user position, whether he or she is an admin or customer. When user position is defined, he or she will be directed to other pages with corresponding functions to the user position.

```
>>> homepage()
```

```
FUNCTION homepage

BEGIN
    DECLARE option AS INTEGER
    PRINT("=== Welcome to Asian Event Management Service (AEMS) ===")
    PRINT("Please select your position:")
    PRINT("1. Admin")
    PRINT("2. Customer")
    PRINT("3. Exit")
    PRINT("Please select an option: ")
    GET option
    IF option = 1 THEN
        CALL FUNCTION adminHome()
    ELSE IF option = 2 THEN
        CALL FUNCTION guestOption()
    ELSE IF option = 3 THEN
        CALL FUNCTION exitPage()
    ELSE
        PRINT("Please select a valid option")
        CALL FUNCTION homepage()
    ENDIF
END
```

To build on the foundation of the `homepage()` function, we first declare the data format of variables. The variable `option` is set to accept only integer values. Next, we print a welcome statement and the available options for users to select their positions. The input should be either `1`, `2`, or `3`. Option 1 is for admins, it will jump to the admin home page by calling the `adminHome()` function; option 2 is for customers, which will call the `guestOption()` function that is designed for customers; lastly, option 3 will jump to `exitPage()` function that is intended to end the

entire program. If any inputs other than `1`, `2`, or `3` is inserted, an error message will be displayed and prompt the user back to the beginning of `homePage()` for reselecting a valid option.



The flowchart of `homepage()` begins with a `BEGIN` in a *terminator* that looks like oval shape. It then goes to the display of welcome message and available options in an *input/output* box. The

diamond shape is a *decision* box, used for user input. A question mark is placed behind the option, generating a question whether the decision was made. Two arrows branch out from `option=1?` to indicate if it is true or false. If user has selected option 1, the output is true therefore function `adminHome()` is called; otherwise, the system will go to the next decision `option=2?`. Similar process occurs in option 2 and option 3. If `option=2?` is true, function `guestOption()` is called; if false, proceeds to `option=3?`. If `option=2?` is true, calls function `exitPage()`; if false, prints an error message and redirect the user back to `homepage()`. *Predefined process* box looks like a rectangle with vertical lines on the left and right, it is used for calling functions. All four possible outputs are linked to a round shape `connecter` and the function closes with `END` in a *terminator*.

Registration System

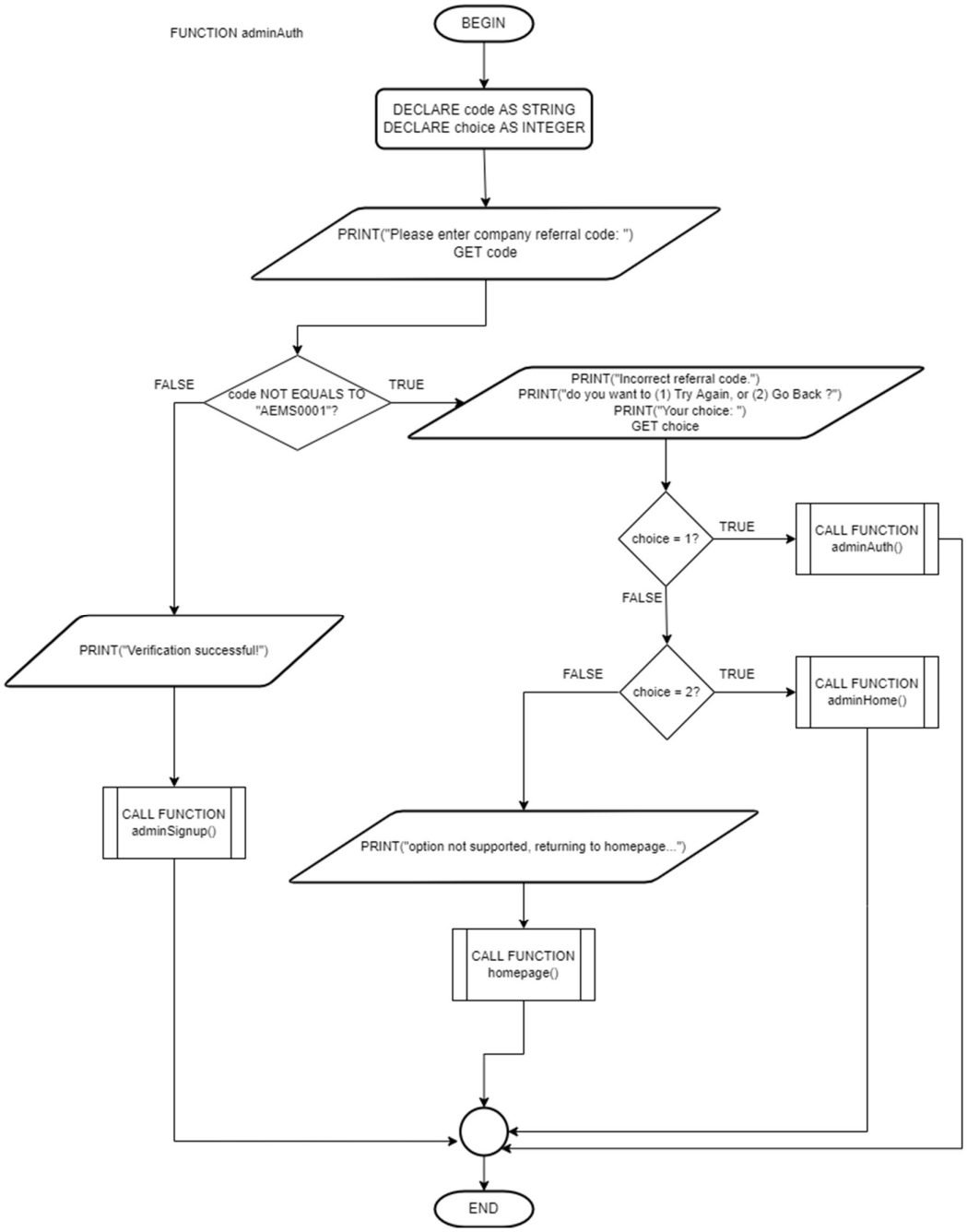
Both admin and customer are required to have an account in the system in order to access most of the functions. The registration system allows users to sign up. There are separate registration systems for admin and member.

```
>>> adminAuth()
```

```
FUNCTION adminAuth

BEGIN
    DECLARE code AS STRING
    DECLARE choice AS INTEGER
    PRINT("Please enter company referral code: ")
    GET code
    IF code NOT EQUALS TO "AEMS0001" THEN
        PRINT("Incorrect referral code.")
        PRINT("do you want to (1) Try Again, or (2) Go Back ?")
        PRINT("Your choice: ")
        GET choice
        IF choice = 1 THEN
            CALL FUNCTION adminAuth()
        ELSE IF choice = 2 THEN
            CALL FUNCTION adminHome()
        ELSE
            PRINT("option not supported, returning to homepage...")
            CALL FUNCTION homepage()
        ENDIF
    ELSE
        PRINT("Verification successful!")
        CALL FUNCTION adminSignup()
    ENDIF
END
```

`adminAuth()` is designed for admins. It verifies admin identity to ensure only system admins are allowed to generate an admin account. This is crucial as to avoid customers from accessing the privilege functions such as modifying event contents in the system. Therefore, this function comes before the signing up of an admin account. AEGIS uses a company referral code to verify user identity, as the code is exclusively known by only admins. The function starts off by defining variable `code` as string and `choice` as integer. The system asks for the referral code through a printed message. The code set is `AEMS0001`, if user inputs the wrong code, the system

notifies user and asks either the user wants to `(1) Try Again` or `(2) Go Back`. The variable used here is `choice`. If user input choice as `1`, system jumps to the start of `adminAuth()`. If choice is equals to 2, it goes to admin home page by calling the `adminHome()` function. If the input is neither `1` nor `2`, the system displays a message telling the user that the option is invalid and proceed to return to `homepage()`. Those are the possible outcomes when user inserted the wrong referral code, however if the user inserted the right code, which is `AEMS0001`, the system prints a verification successful message and directs the user to the `adminSignup()` function for registration.



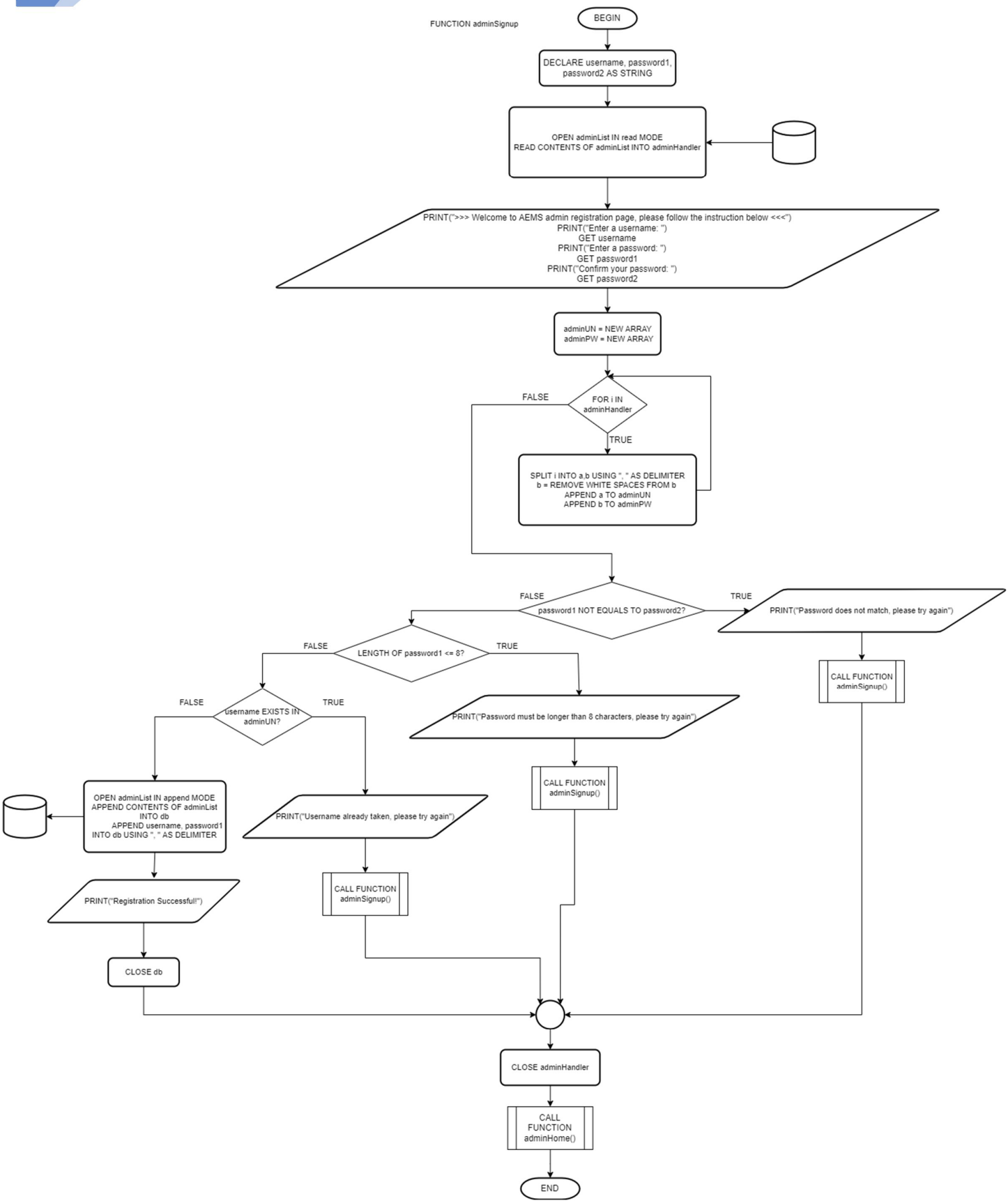
The flowchart of `adminAuth()` starts with a begin in a *terminator* and proceeds to declaring data types of `code` and `choice` in a *process* box. Printing a message to display is an output while obtaining code from user is an input, therefore they are placed in the *input/output* box. When code is obtained, systems decide whether the code is correct using a *decision* box. Note that the condition is `code NOT EQUALS TO "AEMS0001"`. If the condition is true, meaning that user inserted the wrong code and in an *input/output* box, system prints message providing user the choice to either retry or go back. `choice = 1?` and `choice = 2?` are both decisions, therefore each have two branches that stand for `TRUE` and `FALSE`. If choice is equals to 1, the system follows the `TRUE` branch to CALL FUNCTION `adminAuth()`, otherwise, it will proceed to `choice = 2?`. If `choice = 2?` is correct, function `adminHome()` is called; or else, the system prints option unsupported message in *input/output* box and calls `homepage()` function. Alternatively, if the code inserted in the beginning is correct, process box is used for the `print("Verification successful!")`, then calls `adminSignup()`. All possible outputs are connected with a *connecter* and the function ends with the *terminator*.

```
>>> adminSignup()
```

```
FUNCTION adminSignup

BEGIN
    DECLARE username, password1, password2 AS STRING
    OPEN adminList IN read MODE
    READ CONTENTS OF adminList INTO adminHandler
    PRINT(">>> Welcome to AEMS admin registration page, please follow the instruction below <<<")
    PRINT("Enter a username: ")
    GET username
    PRINT("Enter a password: ")
    GET password1
    PRINT("Confirm your password: ")
    GET password2

    adminUN = NEW ARRAY
    adminPW = NEW ARRAY
    FOR i IN adminHandler
        SPLIT i INTO a,b USING ", " AS DELIMITER
        b = REMOVE WHITE SPACES FROM b
        APPEND a TO adminUN
        APPEND b TO adminPW
    ENDFOR

    IF password1 NOT EQUALS TO password2 THEN
        PRINT("Password does not match, please try again")
        CALL FUNCTION adminSignup()
    ELSE IF LENGTH OF password1 <= 8 THEN
        PRINT("Password must be longer than 8 characters, please try again")
        CALL FUNCTION adminSignup()
    ELSE IF username EXISTS IN adminUN THEN
        PRINT("Username already taken, please try again")
        CALL FUNCTION adminSignup()
    ELSE
        OPEN adminList IN append MODE
        APPEND CONTENTS OF adminList INTO db
        APPEND username, password1 INTO db USING ", " AS DELIMITER
        PRINT("Registration Successful!")
        CLOSE db
    ENDIF
    CLOSE adminHandler
    CALL FUNCTION adminHome()
END
```

As the function name indicates, `adminSignup()` is used for registration of admins. First and foremost, `username`, `password1` and `password2` are set as strings. `adminList` is opened in read mode, and the contents are read into the `adminHandler`. A welcome message is printed and prompts user to key in username, password1 and password2. `adminUN` and `adminPW` are defined as new arrays. A *for loop* is used for `i` in `adminHandler`. `i` is split into individual strings `a` and `b` using the `", "` as separator. Empty spaces are removed from `b`. `a` is appended to the adminUN while `b` is appended to `adminUN`, and the loop is closed. There are 3 conditions that must be met to successfully create an account, where `password1` must be equal to `password2`, length of

`password1` must be more than 8, and `username` must not exist in `adminUN`. If either one of these conditions is not satisfied, the system will print an error message and return to the start of `adminSignup()` function. The purpose of doing so is to check the password and username. `password2` is used to confirm `password1` and make sure they match while a minimum of 8 password characters ensures better secured password. Each user must have a unique username so that there is no duplication, duplication will cause errors in the system. If all 3 conditions are met, the system proceed to open the adminList and append `username` and `password1` into the `db` handler. Function `adminHome()` is called and the `adminSignup()` function ends.


FUNCTION adminSignup
BEGIN
DECLARE username, password1, password2 AS STRING
OPEN adminList IN read MODE
READ CONTENTS OF adminList INTO adminHandler
PRINT(">>> Welcome to AEMS admin registration page, please follow the instruction below <<<")
PRINT("Enter a username: ")
GET username
PRINT("Enter a password: ")
GET password1
PRINT("Confirm your password: ")
GET password2
adminUN = NEW ARRAY
adminPW = NEW ARRAY
FOR i IN adminHandler
FALSE
TRUE
SPLIT i INTO a,b USING ", " AS DELIMITER
b = REMOVE WHITE SPACES FROM b
APPEND a TO adminUN
APPEND b TO adminPW
password1 NOT EQUALS TO password2?
FALSE
TRUE
PRINT("Password does not match, please try again")
CALL FUNCTION adminSignup()
LENGTH OF password1 <= 8?
FALSE
TRUE
PRINT("Password must be longer than 8 characters, please try again")
CALL FUNCTION adminSignup()
username EXISTS IN adminUN?
FALSE
TRUE
OPEN adminList IN append MODE
APPEND CONTENTS OF adminList INTO db
APPEND username, password1 INTO db USING ", " AS DELIMITER
PRINT("Username already taken, please try again")
CALL FUNCTION adminSignup()
PRINT("Registration Successful!")
CLOSE db
CLOSE adminHandler
CALL FUNCTION adminHome()
END

Starting with `BEGIN`, the declaring of variables' data types are placed in a *process* box. The opening and reading of `adminList` into `adminHandler` are also placed in a process box, with a *database* going into the process box. Printing welcome message and acquiring username and password are done is *input/output* box. Next is a process box where new arrays are created. For loop is placed in a *decision* box. It proceeds to append `a` and `b` into `adminUN` and `adminPW` and loops back to `for loop` until condition is `FALSE`. When for loop ended, it checks for the 3 conditions for password and username:

1. First condition – `password1 NOT EQUALS TO password2?`
    - If true: take branch to "password does not match" *input/output* box and calls `adminSignup()`.
    - If false: take branch to second condition.
2. Second condition – `LENGTH OF password1 <= 8?`
    - If true: take branch to "password must be longer than 8 characters" *input/output* box and calls `adminSignup()`.
    - If false: take branch to third condition.
3. Third condition – `username EXISTS IN adminUN?`
    - If true: take branch to "username already taken" *input/output* box and calls `adminSignup()`.
    - If false: open `adminList` and append contents into `db` (process box), next prints registration successful message and close the `db`.

Lastly, the final outputs are connected and `adminHandler` is closed. `adminHome()` function is called and the current function ends.

```
>>> memberSignup()
```

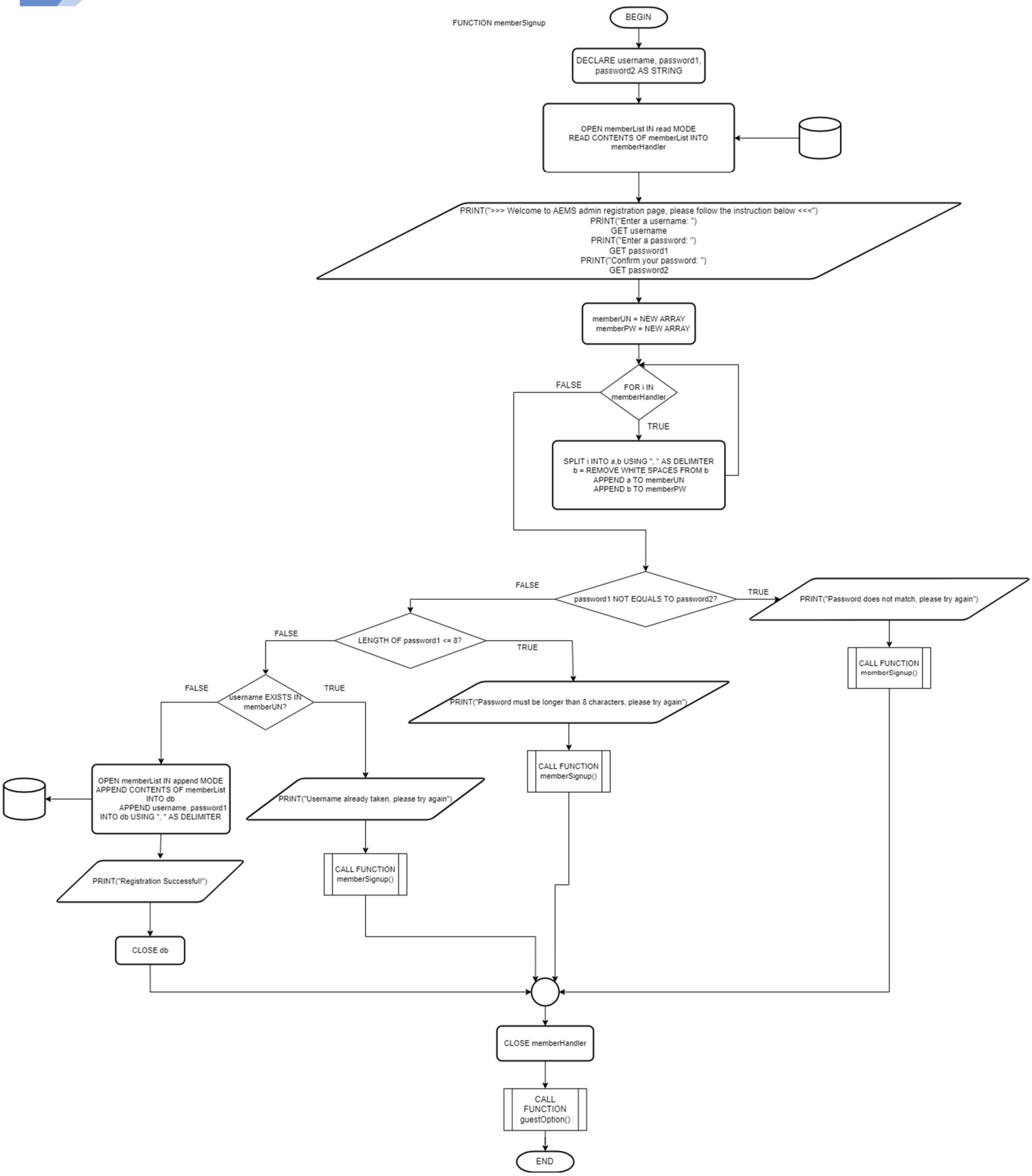
```
FUNCTION memberSignup

BEGIN
    DECLARE username, password1, password2 AS STRING
    OPEN memberList IN read MODE
    READ CONTENTS OF memberList INTO memberHandler
    PRINT(">>> Welcome to AEMS member registration page, please follow the instruction below <<<")
    PRINT("Enter a username: ")
    GET username
    PRINT("Enter a password: ")
    GET password1
    PRINT("Confirm your password: ")
    GET password2

    memberUN = NEW ARRAY
    memberPW = NEW ARRAY
    FOR i IN memberHandler
        SPLIT i INTO a,b USING ", " AS DELIMITER
        b = REMOVE WHITE SPACES FROM  b
        APPEND a TO memberUN
        APPEND b TO memberPW
    ENDFOR

    IF password1 NOT EQUALS TO password2 THEN
        PRINT("Password does not match, please try again")
        CALL FUNCTION memberSignup()
    ELSE IF LENGTH OF password1 <= 8 THEN
        PRINT("Password must be longer than 8 characters, please try again")
        CALL FUNCTION memberSignup()
    ELSE IF username in memberUN THEN
        PRINT("Username already taken, please try again")
        CALL FUNCTION memberSignup()
    ELSE
        OPEN memberList IN append MODE
        APPEND CONTENTS OF memberList INTO db
        APPEND username, password1 INTO db USING ", " AS DELIMITER
        PRINT("Registration Successful!")
        CLOSE db
    ENDIF
    CLOSE memberHandler
    CALL FUNCTION guestOption()
END
```

memberSignup() allows customers to make registrations. It works exactly the same as adminSignup() except that all the “admin” words and admin-related things are switched to member-related, changes are as shown below:

(original from admin) – (edited for member)

- `admin` – `member`
- `adminList` – `memberList`
- `adminHandler` – `memberHandler`
- `adminUN` – `memberUN`

- `adminPW` – `memberPW`
- `adminSignup()` – `memberSignup()`
- `CALL FUNCTION adminHome()` before `END` – `CALL FUNCTION guestOption()` before `END`



Since `memberSignup()` works similarly to `adminSignup()`, the flowchart has no difference.

> Login System

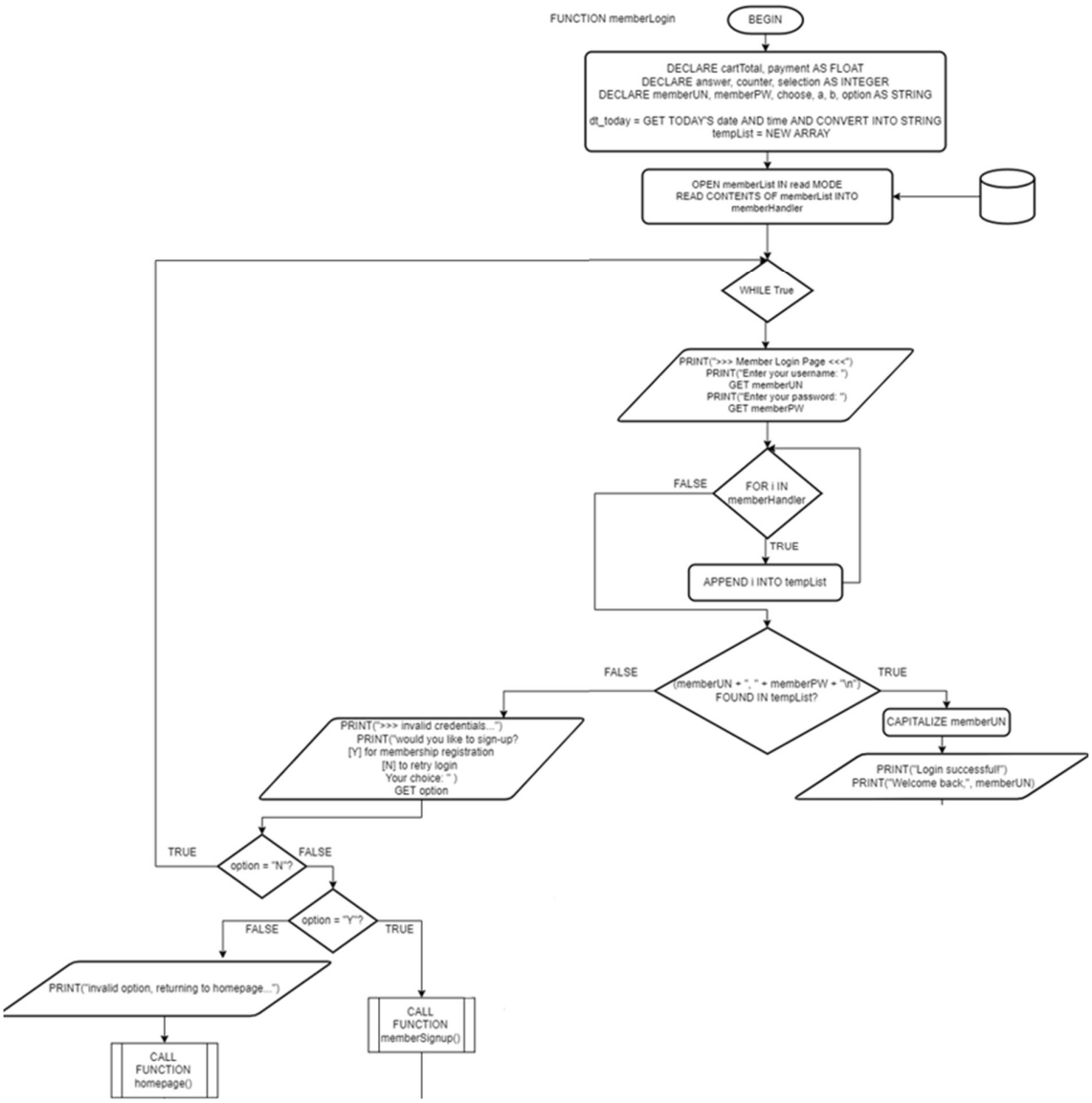
Once successfully signed up an account, users may login to access other functions. The existence of login system allows the system to know who is accessing. In the AEGIS, `memberLogin()` function is extremely long as it included shopping cart and payment systems. Hence, focus is put on only login system under this section. Shopping cart and payment systems will be explained in `Option available for registered customers` section of the document.

```
>>> memberLogin()
```

```
681  FUNCTION memberLogin
682
683  BEGIN
684      DECLARE cartTotal, payment AS FLOAT
685      DECLARE answer, counter, selection AS INTEGER
686      DECLARE memberUN, memberPW, choose, a, b, option AS STRING
687      dt_today = GET TODAY'S date AND time AND CONVERT INTO STRING
688      tempList = NEW ARRAY
689      OPEN memberList IN read MODE
690      READ CONTENTS OF memberList INTO memberHandler
691      WHILE True
692          PRINT(">>> Member Login Page <<<")
693          PRINT("Enter your username: ")
694          GET memberUN
695          PRINT("Enter your password: ")
696          GET memberPW
697          FOR i IN memberHandler
698              APPEND i INTO tempList
699          ENDFOR
700
701          IF (memberUN + ", " + memberPW + "\n") FOUND IN tempList THEN
702              CAPITALIZE memberUN
703              PRINT("Login successful!")
704              PRINT("Welcome back,", memberUN)
```

```
853          ELSE
854              PRINT(">>> invalid credentials...")
855              PRINT("would you like to sign-up?
856              [Y] for membership registration
857              [N] to retry login
858              Your choice: " )
859              GET option
860              IF option = "N" THEN
861                  CONTINUE
862              ELSE IF option = "Y" THEN
863                  CALL FUNCTION memberSignup()
864              ELSE
865                  PRINT("invalid option, returning to homepage...")
866                  CALL FUNCTION homepage()
867              ENDIF
868          ENDIF
869          BREAK
870      ENDWHILE
871      CLOSE memberHandler
872  END
```

The `memberList` will be opened and read into `memberHandler` first. To login, member will need to key in their username and password. The contents in `memberHandler` will be appended into a temporary list, `tempList`. To authenticate the account, the system looks for customer username and password in the format of `memberUN + ", " + memberPW + "\n"` in the `tempList`. If it is found in `tempList`, then login is successful. A welcome message with the member's capitalized username will be printed. In contrast, if the username or password does not exist in the `tempList`, the system will ask if the user wants to register a new account (`Y`) or retry login (`N`). If the user enters `N`, the system loops back to the start of login page where username and password can be inserted. If the user selects `Y`, function `memberSignup()` is called and user will be redirected to the signup page. If user enters something other than `Y` and `N`, system prints a message telling user that it is an invalid option and jumps to the `homepage()` function.



After `BEGIN` and declaring data types, we proceed to open `memberList` and read the contents into `memberHandler`. Proceeding to obtaining username and password from user, contents in memberHandler is appended into `tempList`. Next, if username and password is found in `tempList` in the stated format, login is successful. If not found, system prints message to get `option` from user. If `option = N`, system loops to `WHILE True`; otherwise, goes to second condition. If `option = Y` , `memberSignup()` function is called; otherwise, prints "invalid option" message and calls `homepage()` function.

```
>>> adminLogin()
```

```
FUNCTION adminLogin

BEGIN
    DECLARE adminUN, adminPW, option AS STRING
    tempList = NEW ARRAY
    OPEN adminList IN read MODE
    READ CONTENTS OF adminList INTO adminHandler
    WHILE True
        PRINT(">>> Admin Login Page <<<")
        PRINT("Enter your username: ")
        GET adminUN
        PRINT("Enter your password: ")
        GET adminPW
        FOR i IN adminHandler
            APPEND i TO tempList
        ENDFOR
        IF (adminUN + ", " + adminPW + "\n") FOUND IN tempList THEN
            CAPITALIZE adminUN
            PRINT("Login successful!")
            PRINT("Welcome back,", adminUN)
            CALL FUNCTION adminOption()
            BREAK
        ELSE
            PRINT("
            >>> invalid credentials...
            ")
            PRINT("would you like to sign-up?
            [Y] for admin registration
            [N] to retry login
            Your choice: ")
            GET option
            IF option = "N" THEN
                CONTINUE
            ELSE IF option = "Y" THEN
                CALL FUNCTION adminAuth()
            ELSE
                PRINT("invalid option, returning to homepage...")
                BREAK
            ENDIF
        ENDIF
    ENDWHILE
    CLOSE adminHandler
    CALL FUNCTION homepage()
END
```

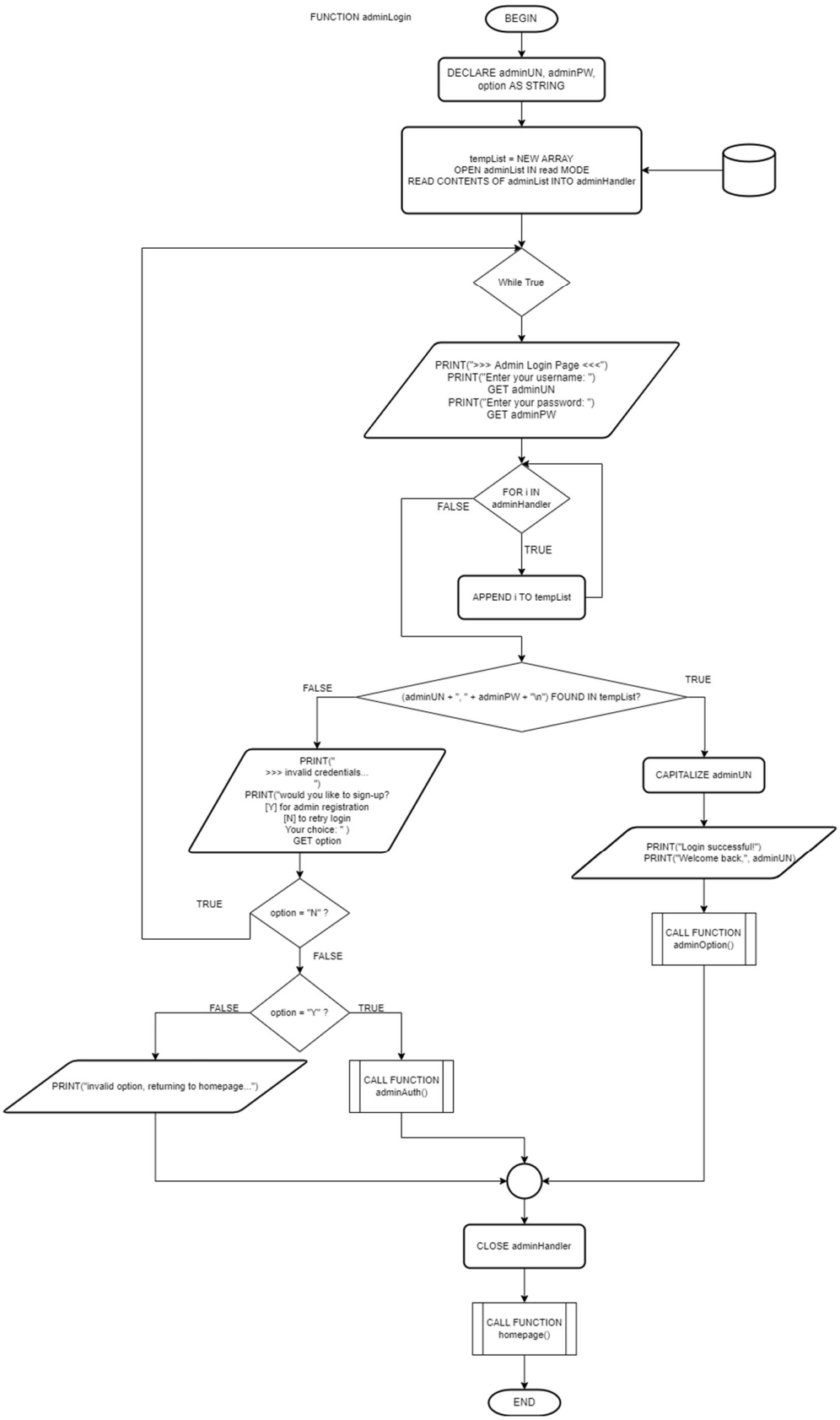
The pseudocode for `adminLogin()` is same as memberLogin(), except that the word “member” is converted to “admin”. Changes are as shown:

(original from member) – (new for admin)

- `Member` - `Admin`
- `memberUN` – `adminUN`
- `memberPW` – `adminPW`

- `memberList` – `adminList`
- `memberHandler` – `adminHandler`

Besides that, when the user opts for admin registration after failing to login (`line-166`), he or she will be redirected to `adminAuth()` function instead of `adminSignup()`, unlike member which calls the `memberSignup()` directly. This is due to the fact that admins require authentication before signing up.



The flowchart of `adminLogin()` works similarly to `memberLogin()`.

> Options available for all customers

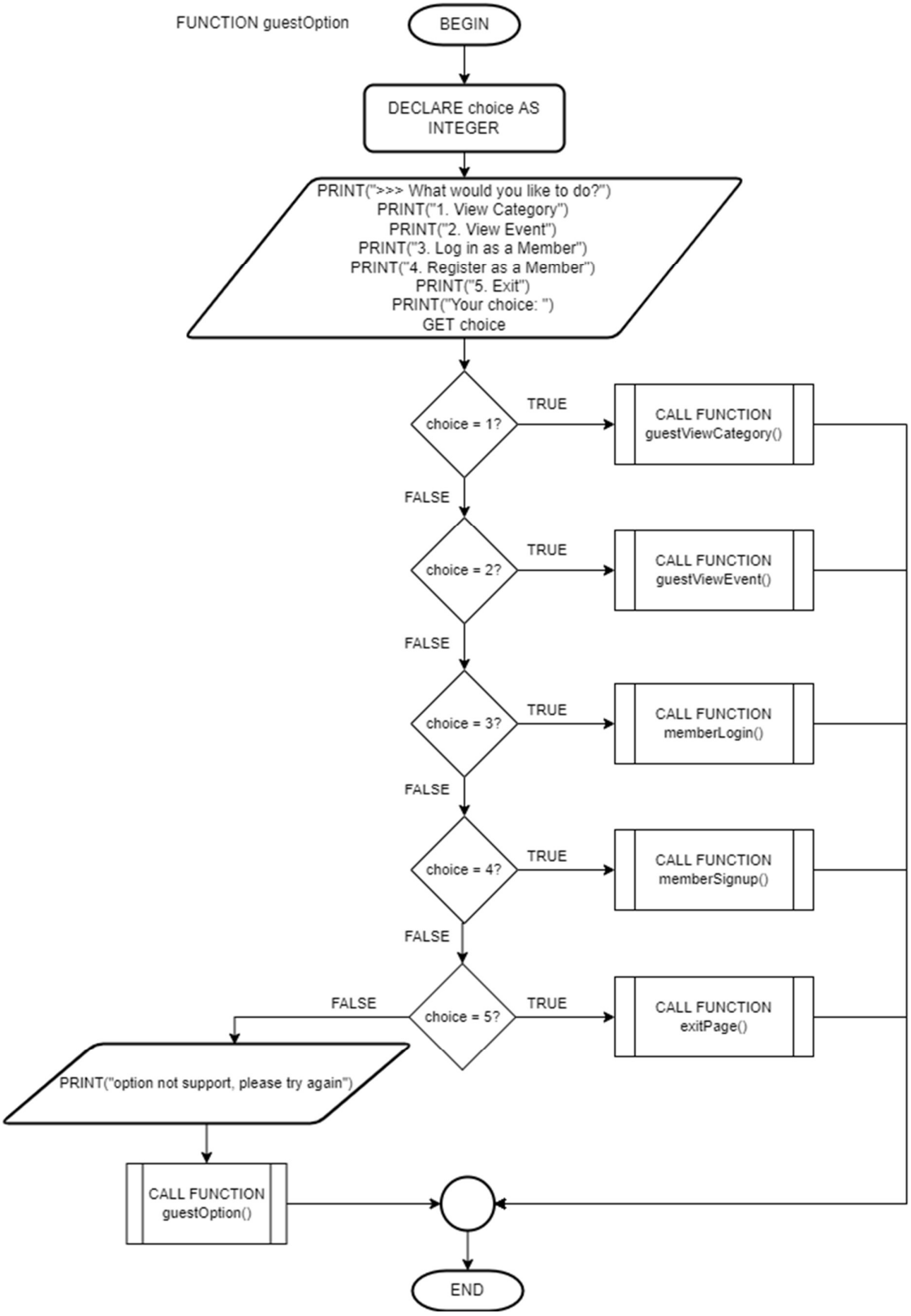
Customer login is required to add events to cart and make purchases. The unregistered customers, however, can access some of the functions in AEGIS. `guestOption()`, `guestViewCategory()` and `guessViewEvent()` can be accessed by all customers.

`>>> guestOption()`

```
598  FUNCTION guestOption
599
600  BEGIN
601      DECLARE choice AS INTEGER
602      PRINT(">>> What would you like to do?")
603      PRINT("1. View Category")
604      PRINT("2. View Event")
605      PRINT("3. Log in as a Member")
606      PRINT("4. Register as a Member")
607      PRINT("5. Exit")
608      PRINT("Your choice: ")
609      GET choice
610      IF choice = 1 THEN
611          CALL FUNCTION guestViewCategory()
612      ELSE IF choice = 2 THEN
613          CALL FUNCTION guestViewEvent()
614      ELSE IF choice = 3 THEN
615          CALL FUNCTION memberLogin()
616      ELSE IF choice = 4 THEN
617          CALL FUNCTION memberSignup()
618      ELSE IF choice = 5 THEN
619          CALL FUNCTION exitPage()
620      ELSE
621          PRINT("option not support, please try again")
622          CALL FUNCTION guestOption()
623      ENDIF
624  END
```

`guestOption()` provides 5 features that users may access, `1` to view all categories, `2` to view all events, `3` to log in, `4` to register account and `5` to exit the program. User input is named as `choice`, and is set as `integer`. There are six possible outputs depending on user's choice:

- if `choice` is `1`: goes to `guestViewCategory()`
- if `choice` is `2`: goes to `guestViewEvent()`
- if `choice` is `3`: goes to `memberLogin()`
- if `choice` is `4`: goes to `memberSignup()`
- if `choice` is `5`: goes to `exitPage()`
- if user input is none of the above choices, option not supported messaged is printed and system loops back to `guestOption()` again



In the flowchart of memberOption(), once `choice` is declared as integer, system prints output message and gets user `choice`. System checks the conditions in the following sequence:

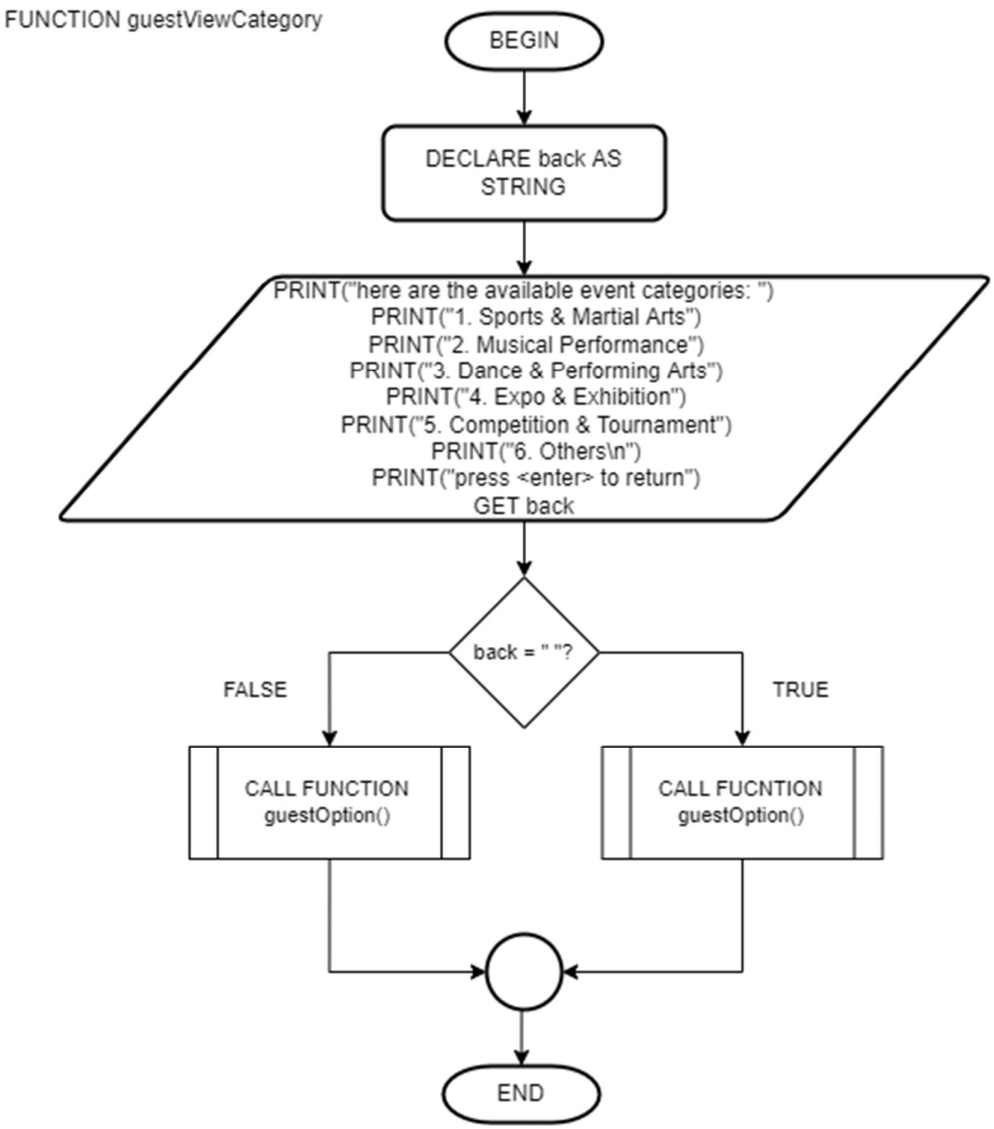
- condition 1 – `choice = 1?`
  if true: branches to call `guestViewCategory()`
  if false: goes to condition 2
- condition 2 – `choice = 2?`
  if true: branches to call `guestViewEvent()`
  if false: goes to condition 3
- condition 3 – `choice = 3?`
  if true: branches to call `memberLogin()`
  if false: goes to condition 4
- condition 4 – `choice = 4?`
  if true: branches to call `memberSignup()`
  if false: goes to condition 5
- condition 5 – `choice = 5?`
  if true: branches to call `exitPage()`
  if false: prints message and calls `guestOption()`

All outcomes link to a connector and function ends.

```
>>> guestViewCategory()
```

```
FUNCTION guestViewCategory

BEGIN
    DECLARE back AS STRING
    PRINT("here are the available event categories: ")
    PRINT("1. Sports & Martial Arts")
    PRINT("2. Musical Performance")
    PRINT("3. Dance & Performing Arts")
    PRINT("4. Expo & Exhibition")
    PRINT("5. Competition & Tournament")
    PRINT("6. Others\n")
    PRINT("press <enter> to return")
    GET back
    IF back = "" THEN
        CALL FUCNTION guestOption()
    ELSE
        CALL FUNCTION guestOption()
    ENDIF
END
```

`guestViewCategory()` allows all customers to view the available event categories. The system prints all categories only for viewing purposes. Therefore, despite any inputs given by user, the system will return to guest option page by calling `guestOption()`. `back` is set as the variable for inputs. If nothing is inserted and *enter* is clicked, it is an empty string `“”`. `guestOption()` is called not matter the string is empty or not.



The flow of `guestViewCategory()` is as shown above. After declaring data type, the system prints all event categories. If `back = “”?` is true, `guestOption()` is called; if false, `guestOption()` is also called. Results are connected and function ends.

```
>>> guestViewEvent()
```

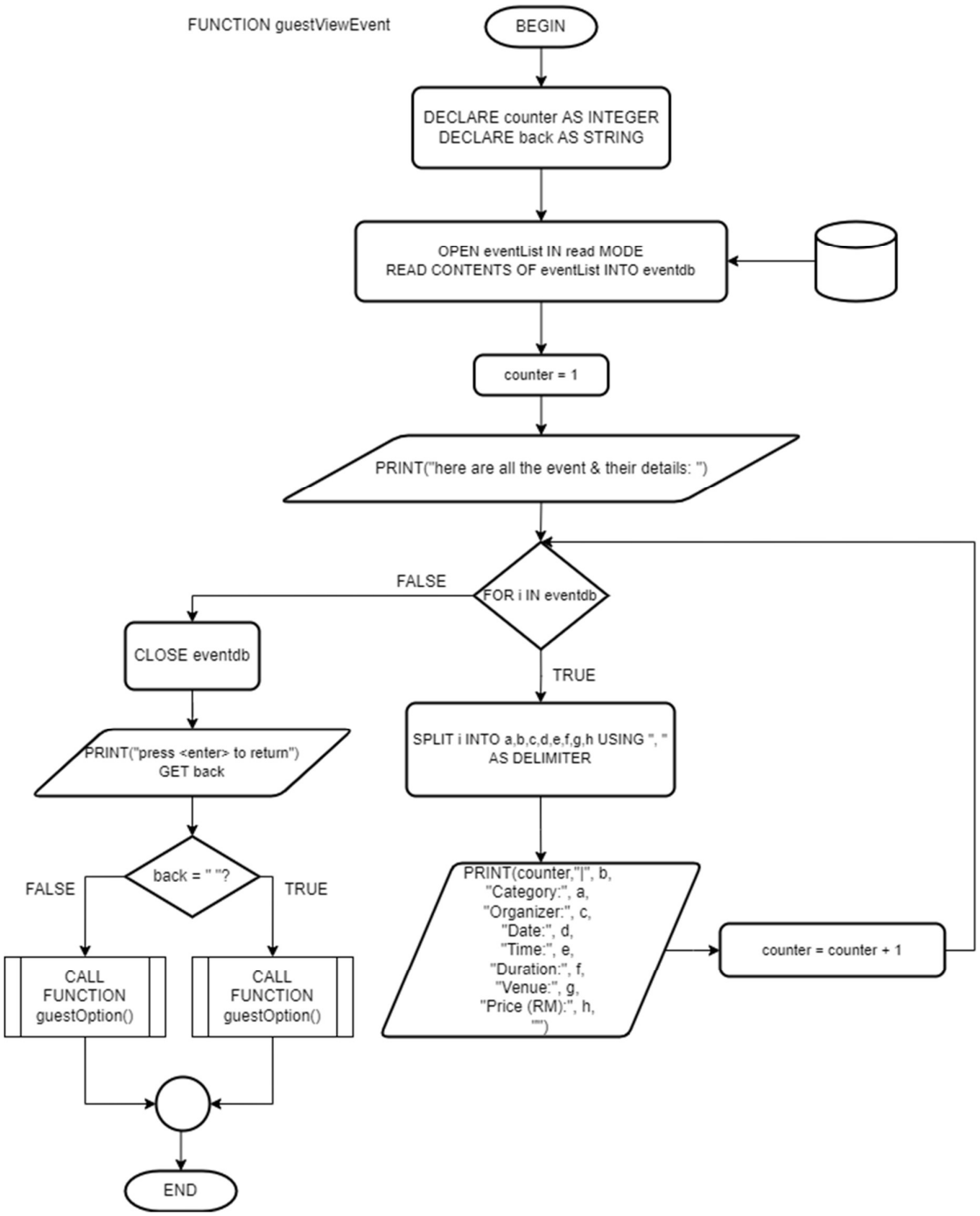
```
FUNCTION guestViewEvent

BEGIN
    DECLARE counter AS INTEGER
    DECLARE back AS STRING
    OPEN eventList IN read MODE
    READ CONTENTS OF eventList INTO eventdb
    counter = 1
    PRINT("here are all the event & their details: ")
    FOR i IN eventdb
        SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
        PRINT(counter,"|", b,
        "Category:", a,
        "Organizer:", c,
        "Date:", d,
        "Time:", e,
        "Duration:", f,
        "Venue:", g,
        "Price (RM):", h,
        "")
        counter = counter + 1
    ENDFOR
    CLOSE eventdb
    PRINT("press <enter> to return")
    GET back
    IF back = "" THEN
        CALL FUNCTION guestOption()
    ELSE
        CALL FUNCTION guestOption()
    ENDIF
END
```

`guestViewEvent()` allows customers to view all available events. Accessing and reading of the `eventList` is required in this case. `eventdb` is the handler used to read `eventList` and temporarily store the event contents. Each event with its details is stored as a string in the `eventList`, therefore each string (`i`) will be extracted and split into smaller strings, one string per event detail. `i` is split into 8, namely `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`. Initially in the `i`, `“, ”` is placed between each event detail and everything combines into a single string. Therefore, AEGIS will now use the `“, ”` as the separator. When splitting is done, we now have 8 strings and each of them will be printed accordingly.

- `b` is set as event name
- `a` is set as event category
- `c` is set as event organizer
- `d` is set as event date
- `e` is set as event time
- `f` is set as event duration

- `g` is set as event venue
- `h` is set as. event price

The entire set of strings, from `a` to `h`, are event details that belong to one event, and each event detail is printed on a separate line. To further display all events, the process will need to iterate. Therefore, we use `counter` and `for loop`. Each time the process loops, the counter value increases by 1, it is the numbering value that defines which string to process next, so that all events can be displayed. The `counter` is also printed right next to `b` (`line-659`) for neatness of the printed message while allowing user to know how many events are available. When all events are displayed, `for loop` ends and `eventdb` is closed. Since this function only allows viewing of events, system will return to `guestOption()` no matter what input it receives from users.



After declaring data types, opening and reading `eventList`, the for loop begins with `counter = 1`. Using the `TRUE` branch, the first string (`i`) is split into `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`. These newly split strings, which are event details, are included in the print function for display. The `counter` then increases by 1 and goes to top of `for loop` for the second round of splitting and printing. The entire process iterates until all `i` in the `eventdb` has been split and printed. Systems proceeds to the `FALSE` branch and close the `eventdb`. If user inputs *nothing*, meaning `back` is equal to `“”` , function `guestOption()` is called; while any other inputs given will also lead to `guestOption()`.

> Option available for registered customers

Some functions are exclusively available to registered customers, such as creating shopping carts, making payments and viewing registered events. These options are included under `memberLogin()` which was not mentioned earlier.

```
>>> memberLogin()
```

now talk about the shopping cart, view events, and exit features

how they work (cart features, display event algorithm [1,0,1,0,0,0] , etc)

```
705             memberName = memberUN
706             cartItems = NEW ARRAY
707             cartTotal = 0
708
709             WHILE True
710                 PRINT("\nwhat would you like?")
711                 PRINT("1. Add events to Shopping Cart")
712                 PRINT("2. View purchased events")
713                 PRINT("3. Exit")
714                 PRINT("Your Choice: ")
715                 GET answer
```

After successfully logging in, customers have 3 options to select from, which are add event to carts, view purchased events and exit the program. Variable used here is `answer`.

If `1` is inputted, the following procedure takes place.

```
717             IF answer = 1 THEN
718                 WHILE True
719                     counter = 1
720                     PRINT("Which event would you like to add?")
721                     OPEN eventList IN read MODE
722                     READ CONTENTS OF eventList INTO eventdb
723                     FOR i IN eventdb
724                         SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
725                         PRINT(counter,"|", b)
726                         counter = counter + 1
727                     ENDFOR
728                     CLOSE eventdb
729
730                     PRINT("Your choice: ")
731                     GET selection
732                     CHECK (selection IS EQUAL TO DATATYPE integer) THEN
733                         IF selection IS FROM 0 TO counter THEN
734                             selection = selection - 1
735                             PASS
736                         ELSE
737                             PRINT("\n>>> selected value out of bond, try again\n")
738                             CONTINUE
739                     ELSE
740                         PRINT("\n>>> please insert an integer value\n")
741                         CONTINUE
742                     END CHECK
```

System asks which event the customer wants to add to cart and print all available events. To display available events, the `eventList` needs to be opened and read by the `eventdb`. The string in `eventdb` will be split into smaller strings. According to the split sequence, `b` is the event name, therefore only content of `b` is printed. This process prints only one event name, therefore iteration is needed to print following events. `counter` increases by 1 and entire process loops. When all events are printed, the loop ends and close `eventdb`. Now that all events can be viewed by customer, system asks for user's choice. The user input is set as variable `selection` which is set to be `integer`. If integer is inserted, system checks whether the integer value is within 0 to the counter value, the counter value is the final number before the for loop ended. If `selection` is within the range, the system proceeds to minus 1 from `selection` value. This is because strings begin from array location 0. For instance, if user selected first event, he or she will input `1`; in the system however, the first string is stored at array location 0, therefore system needs to minus 1 to get the correct event. On the other hand, if customer inserted an integer but it is out of the available range, system prints error message and loops back to let user re-enter a new value. While if user input is not of data type integer, such as string and float, error message will also be printed and loops back to allow user to insert a new choice.

```
744                         OPEN eventList IN read MODE
745                         READ CONTENTS OF eventList INTO itemdb
746                         COPY CONTENTS FROM itemdb TO itemList
747                         itemIndex = SPLIT itemList AT LOCATION selection USING "," AS DELIMITER
748                         addItem = COPY CONTENT FROM itemIndex AT ARRAY LOCATION 1
749                         itemPrice = REPLACE '\n' AS '' FROM itemIdex AT ARRAY LOCATION 7
750                         itemPrice = CONVERT itemPrice INTO FLOAT
751                         CLOSE itemdb
752
753                         APPEND addItem INTO cartItems
754                         cartTotal = cartTotal + itemPrice
755                         PRINT("do you want to add more items? <Y/N>")
756                         GET choose
757                         IF choose = "Y" THEN
758                             CONTINUE
759                         ELSE IF choose = "N" THEN
760                             PRINT("proceeding to payment...")
761                             BREAK
762                         ELSE
763                             PRINT("option not supported, proceeding to payment")
764                             BREAK
765                         ENDIF
766                     ENDWHILE
```

Next the system opens `eventList` and reads the contents into `itemdb`. The contents in `itemdb` is then copied to `itemList`. Next, we split the contents of `selection` from the `itemList` using `", "` and save it as `itemIndex`. Now that each detail of the event chosen by user (`selection`) is an individual string, but we want to acquire only the name and price of event. After splitting, the

name of event will be located at `array location 1` in the `itemIndex`, therefore we copy it and save it as `addItem`. The price of event is located at `array location 7` and is currently a string, however it will used for calculation is payment later. Therefore, we removed the `\n` which was previously added to the string for neatness (starts a new line), by replacing it with `''`, and proceed to save it as `itemPrice`. Next, we convert data type of `itemPrice` from string to float, and close the `itemdb`. The `addItem` is then appended to `cartItems` to add the event into cart. The total price for payment is named as `cartTotal`, and each time a new event is added to cart, the `itemPrice` is added to `cartTotal`. The system then asks if the customer wants to add another event into cart. If customer inserts `Y`, the system loops back to `line-718` to display available events; if customer inserts `N`, system proceeds to the next section, which is payment; while if user inserts something other than `Y` and `N`, system notifies user that the option is not supported and proceeds to the payment section. The following procedure is for payment system.

```
768                 WHILE True
769                     PRINT("total amount dued: ", cartTotal)
770                     PRINT("please enter your payment amount: ")
771                     GET payment
772                     CHECK (payment IS EQUAL TO DATATYPE float) THEN
773                         IF payment = cartTotal THEN
774                             PRINT("payment successful!")
775                             summaryPrice = cartTotal
776                             balance = 0.00
777                             BREAK
778                         ELSE IF payment < cartTotal THEN
779                             PRINT("insufficient amount, please try again")
780                             CONTINUE
781                         ELSE IF payment > cartTotal THEN
782                             balance = payment - cartTotal
783                             PRINT("payment successful! RM", balance, "have been returned to your account")
784                             summaryPrice = cartTotal
785                             BREAK
786                         ELSE
787                             PRINT("please insert a valid number")
788                             CONTINUE
789                         ENDIF
790                     ELSE
791                         PRINT("\n>>> please insert a valid amount!\n")
792                         CONTINUE
793                     END CHECK
794                 ENDWHILE
795
796                 PRINT("Order Summary:")
797                 PRINT("You bought:", cartItems)
798                 PRINT("Total cost:", cartTotal)
799                 PRINT("You paid:", payment)
800                 PRINT("Balance returned:", balance)
801                 PRINT("order completed! thank you for your support!")
802                 PRINT('')
803                 CLOSE eventdb
```

First and foremost, the total price of events will be printed. User are required to input the amount they are be paying, saved as `payment` variable. The data format of `payment` must be float, therefore if customer enters characters, the system returns to `line-768` for user to re-enter

payment amount. If user inputs correct data type, system checks if the payment amount is equivalent to `cartTotal`, if yes, payment is successful `summaryPrice` and `balance` is generated for order summary which will be printed later. If payment amount is lesser than `cartTotal`, system loops back to `line-768` for user to repay. If payment amount is more than `cartTotal`, `balance` will be return to customer account. `balance` can be obtained with `payment` deducting `cartTotal`. Order summary is printed, which contains `cartItem`, `cartTotal`, `payment`, `balance`, and thank you message. `eventdb` is closed.

```
805                 OPEN cartList IN append MODE
806                 APPEND CONTENTS OF cartList INTO cartdb
807                 FOR i IN cartItems
808                     WRITE (memberName+", "+i+", "+"paid on "+dt_today+"\n") INTO cartdb
809                 ENDFOR
810                 CLOSE cartdb
811                 CONTINUE
```

Lastly, to store the order into system, `cartList` in opened and appended into `cartdb`. In a `for loop`, the `cartItems` will be stored as the format shown in above screenshot (`line-808`). The loop ends when all contents have been stored. `cartdb` is closed, and the system loops back to `line-709` for user to select other options, including exit the program.

Continuing from `line-709`, if option `2` was chosen, the following procedure occurs.

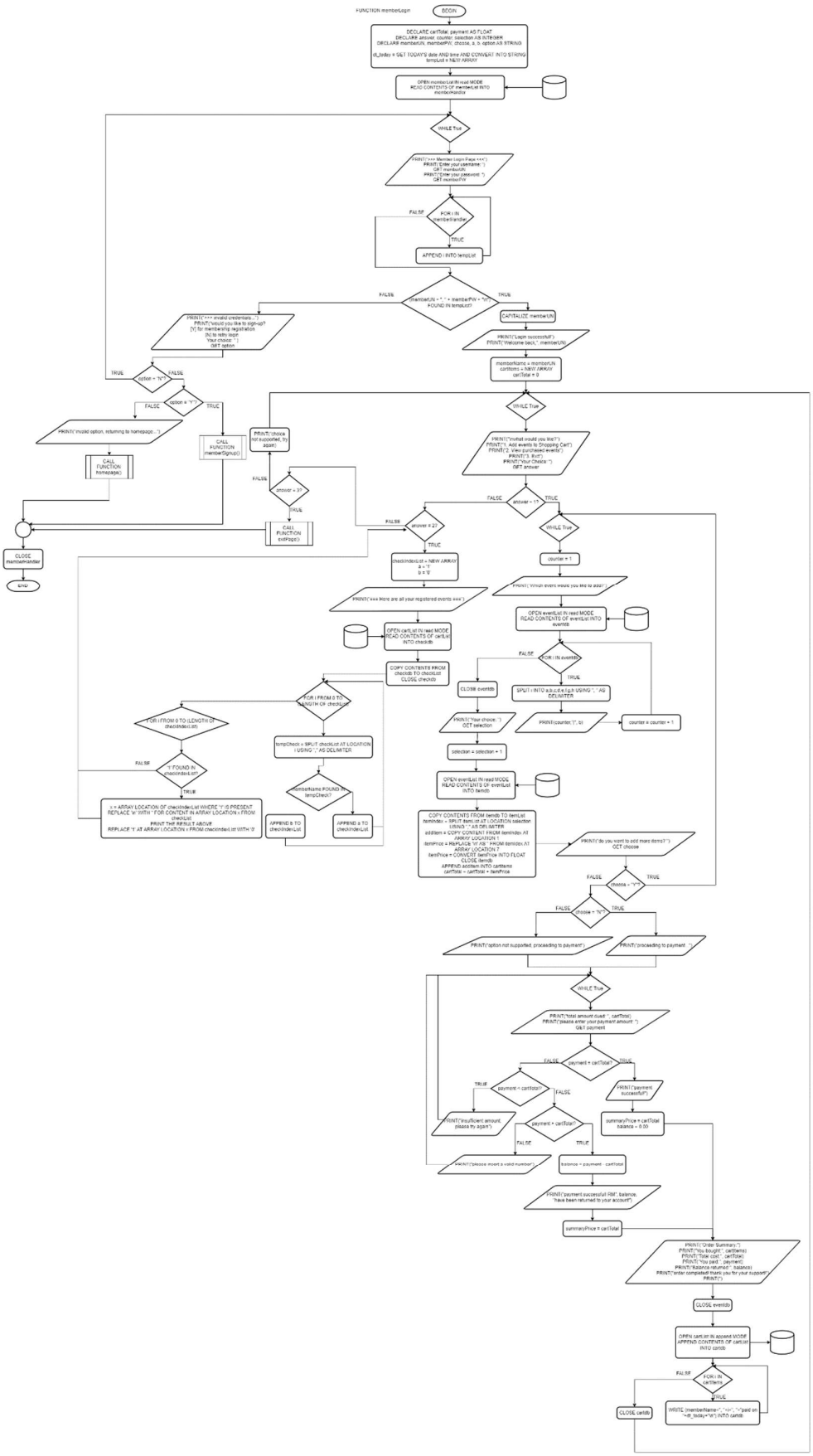
```
814                 ELSE IF answer = 2 THEN
815                     checkIndexList = NEW ARRAY
816                     a = '1'
817                     b = '0'
818                     PRINT("=== Here are all your registered events ===")
819                     OPEN cartList IN read MODE
820                     READ CONTENTS OF cartList INTO checkdb
821                     COPY CONTENTS FROM checkdb TO checkList
822                     CLOSE checkdb
823                     FOR i FROM 0 TO (LENGTH OF checkList)
824                         tempCheck = SPLIT checkList AT LOCATION i USING "," AS DELIMITER
825                         IF memberName FOUND IN tempCheck THEN
826                             APPEND a TO checkIndexList
827                         ELSE
828                             APPEND b TO checkIndexList
829                         ENDIF
830                     ENDFOR
831
832                     FOR i FROM 0 TO (LENGTH OF checkIndexList)
833                         IF '1' FOUND IN checkIndexList THEN
834                             x = ARRAY LOCATION OF checkIndexList WHERE '1' IS PRESENT
835                             REPLACE '\n' WITH '' FOR CONTENT IN ARRAY LOCATION x FROM checkList
836                             PRINT THE RESULT ABOVE
837                             REPLACE '1' AT ARRAY LOCATION x FROM checkIndexList WITH '0'
838                         ELSE
839                             PASS
840                         ENDIF
841                     ENDFOR
842                     CONTINUE
843
844                 ELSE IF answer = 3 THEN
845                     CALL FUNCTION exitPage()
846
847                 ELSE
848                     PRINT("choice not supported, try again)
849                     CONTINUE
850                 ENDIF
851                 BREAK
```

First of all, a new array `checkIndexList` is created and `a` is defined as `‘1’`, `b` is defined as `‘0’`. To view registered events, system needs to open the `cartList` and read the contents into `checkdb`. The contents in `checkdb` is copied to a temporary `checkList`. For all strings in the `checkList`, they are split using “`, `” and saved as `tempCheck`. If the `memberName` exists in `tempCheck`, it means there is record of the user registering for event, to indicate that, `a` is appended to `checkIndedList`. If `memberName` is not found, then `b` is appended to the `checkIndexList`. The entire process loops until all strings have been checked, the `checkIndexList` will contain ones and zeros like `‘1’,‘0’,‘0’,‘1’,‘1’,‘0’,‘1’`, where `‘1’` indicates record exists and `‘0’` indicates otherwise, from this example, string 1, 5, 6 and 7 is `‘1’`, therefore user register records exists in these four strings. Next, to extract the registered event details, another `for loop` is used to check the `checkIndexList`. If `‘1’` is found in the list,

the array location of the `'1'` from `checkIndexList` is saved as `x`. Now we can extract the member's registered event information from `checkList` as we know the array location where data exists. The `\n` will be removed from the content located at array location `x` in the `checkList` and the content is printed. Since the content has been printed, we do not want duplication therefore we replace `'1'` with `'0'` for printed events. While if '1' is not found in `checkIndexList`, the system loops back to check if the next string is `'1'`. The process repeats until all the contents in `checkIndexList` is `'0'`, meaning all strings that belong to the member who registered for events have already been printed. The procedure for checking registered events end here and the system loops back to `line-709` for user to select other options.

If user selects option `3` from `line-709`, the system calls `exitPage()` function to exit the program.

If user inputs something other than `1`, `2`, `3`, error message is printed and system loops back to `line-709`.



> Option for admins

Admins have the options to add and modify events as well as displaying and searching records.

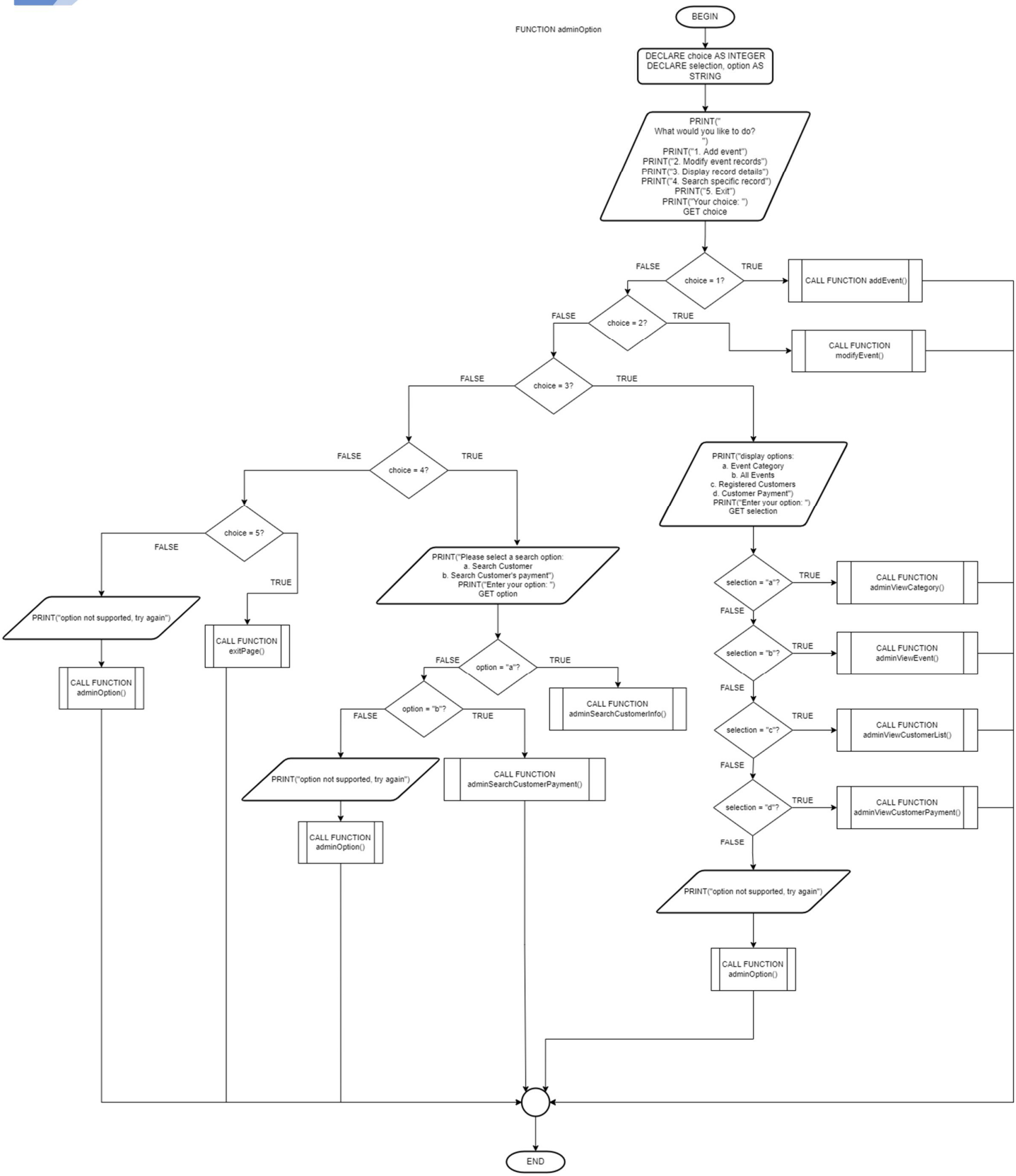
```
>>> adminOption()
```

```
FUNCTION adminOption

BEGIN
    DECLARE choice AS INTEGER
    DECLARE selection, option AS STRING
    PRINT("
    What would you like to do?
    ")
    PRINT("1. Add event")
    PRINT("2. Modify event records")
    PRINT("3. Display record details")
    PRINT("4. Search specific record")
    PRINT("5. Exit")
    PRINT("Your choice: ")
    GET choice
    IF choice = 1 THEN
        CALL FUNCTION addEvent()
    ELSE IF choice = 2 THEN
        CALL FUNCTION modifyEvent()
    ELSE IF choice = 3 THEN
        PRINT("display options:
        a. Event Category
        b. All Events
        c. Registered Customers
        d. Customer Payment")
        PRINT("Enter your option: ")
        GET selection
        IF selection = "a" THEN
            CALL FUNCTION adminViewCategory()
        ELSE IF selection = "b" THEN
            CALL FUNCTION adminViewEvent()
        ELSE IF selection = "c" THEN
            CALL FUNCTION adminViewCustomerList()
        ELSE IF selection = "d" THEN
            CALL FUNCTION adminViewCustomerPayment()
        ELSE
            PRINT("option not supported, try again")
            CALL FUNCTION adminOption()
        ENDIF
    ELSE IF choice = 4 THEN
        PRINT("Please select a search option:
        a. Search Customer
        b. Search Customer's payment")
        PRINT("Enter your option: ")
        GET option
        IF option = "a" THEN
```

```
            CALL FUNCTION adminSearchCustomerInfo()
        ELSE IF option = "b" THEN
            CALL FUNCTION adminSearchCustomerPayment()
        ELSE
            PRINT("option not supported, try again")
            CALL FUNCTION adminOption()
        ENDIF
    ELSE IF choice = 5 THEN
        CALL FUNCTION exitPage()
    ELSE
        PRINT("option not supported, try again")
        CALL FUNCTION adminOption()
    ENDIF
END
```

The `adminOption()` contains general features that admins can access. There are 5 features that admin can select from. `1` for adding event, `2` for modifying event, `3` for displaying records, `4` for searching records and `5` for exiting the AEGIS program. Options are as shown:

- select 1, `addEvent()` function is called
- select 2, `modifyEvent()` function is called
- select 3, options for displaying feature is printed. Options contain `a` (event category), `b` (all events), `c` (registered customers), `d` (customer payment).
    - If selects a, `adminViewCategory()` is called.
    - If selects b, `adminViewEvent()` is called.
    - If selects c, `adminViewCustomerList()` is called.
    - If selects d, `adminViewCustomerPayment()` is called.
- select 4, options for searching record is printed. Options contain `a` (search for customer), `b` (search specific customer record).
    - If selects a, `adminSearchCustomerInfo()` is called.
    - If selects b, `adminSearchCustomerPayment()` is called.
- select 5, `exitPage()` function is called.
- if input is none of the above selections, `adminOption()` is called.


FUNCTION adminOption
BEGIN
DECLARE choice AS INTEGER
DECLARE selection, option AS STRING
PRINT("
What would you like to do?
")
PRINT("1. Add event")
PRINT("2. Modify event records")
PRINT("3. Display record details")
PRINT("4. Search specific record")
PRINT("5. Exit")
PRINT("Your choice: ")
GET choice
choice = 1?
FALSE
TRUE
CALL FUNCTION addEvent()
choice = 2?
CALL FUNCTION modifyEvent()
choice = 3?
choice = 4?
choice = 5?
PRINT("option not supported, try again")
CALL FUNCTION adminOption()
CALL FUNCTION exitPage()
PRINT("display options:
a. Event Category
b. All Events
c. Registered Customers
d. Customer Payment")
PRINT("Enter your option: ")
GET selection
selection = "a"?
CALL FUNCTION adminViewCategory()
selection = "b"?
CALL FUNCTION adminViewEvent()
selection = "c"?
CALL FUNCTION adminViewCustomerList()
selection = "d"?
CALL FUNCTION adminViewCustomerPayment()
PRINT("option not supported, try again")
CALL FUNCTION adminOption()
PRINT("Please select a search option:
a. Search Customer
b. Search Customer's payment")
PRINT("Enter your option: ")
GET option
option = "a"?
CALL FUNCTION adminSearchCustomerInfo()
option = "b"?
PRINT("option not supported, try again")
CALL FUNCTION adminSearchCustomerPayment()
CALL FUNCTION adminOption()
END

The flowchart of adminOption() is as shown above. After obtaining the variable `choice`, which is input from admins, system proceeds to the decision box.

- Condition 1 – `choice = 1?`
  If true: calls addEvent()
  If false: goes to condition 2
- Condition 2 – `choice = 2?`
  If true: calls modifyEvent()
  If false: goes to condition 3
- Condition 3 – `choice = 3?`
  If true: prints display options
  - Condition 1 – `selection = "a"?`
    If true: calls adminViewCategory()
    If false: goes to condition 2
  - Condition 2 – `selection = "b"?`
    If true: calls adminViewEvent()
    If false: goes to condition 3
  - Condition 3 – `selection = "c"?`
    If true: calls adminViewCustomerList()
    If false: goes to condition 4
  - Condition 4 – `selection = "d"?`
    If true: calls adminViewPayment()
    If false: prints option unsupported message and calls adminOption()

  If false: goes to condition 4
- Condition 4 – `choice = 4?`
  If true: prints search options
  - Condition 1 – `option = "a"?`
    If true: calls adminSearchCustomerInfo()
    If false: goes to condition 2
  - Condition 2 – `option = "b"?`
    If true: calls adminSearchCustomerPayment()
    If false: prints option unsupported message and calls adminOption()

  If false: goes to condition 5
- Condition 5 – `choice = 5?`

If true: calls exitPage()

If false: prints option unsupported message and calls adminOption()

event options:

`>>> addEvent() extension`

```
BEGIN
    DECLARE choice, option AS INTEGER
    DECLARE eventName, eventOrg, eventDate, eventTime, eventDuration, eventLocation, eventFee, eventPrice, newCat AS STRING
    OPEN eventList IN read MODE
    READ CONTENTS OF eventList INTO eventHandler
    PRINT(">>> Welcome to AEMS event page, please select your category of events to add <<<")
    WHILE True
        PRINT("Event Category:
        1. Sports/Martial Arts
        2. Musical Performance
        3. Dance/Performing Arts
        4. Expo/Exhibition
        5. Competition/Tournament
        6. Others")
        PRINT("Please enter your choice: ")
        GET choice
```

To add event, the system first asks which category the new event belongs to. 6 choices are given.

```
        IF choice = 1 THEN
            eventCat = "Sports & Martial Arts"
            BREAK
        ELSE IF choice = 2 THEN
            eventCat = "Musical Performance"
            BREAK
        ELSE IF choice = 3 THEN
            eventCat = "Dance & Performing Arts"
            BREAK
        ELSE IF choice = 4 THEN
            eventCat = "Expo & Exhibition"
            BREAK
        ELSE IF choice = 5 THEN
            eventCat = "Competition & Tournament"
            BREAK
        ELSE IF choice = 6 THEN
            PRINT("Enter the name of your category: ")
            GET newCat
            eventCat = newCat
            BREAK
        ELSE
            PRINT("please select a valid option")
            CONTINUE
        ENDIF
    ENDWHILE
```

If admin selects 1, `eventCat` is set as `“Sports & Martial Arts”`, if selects 2, `eventCat` is set as `“MusicalPerformance”` and so on until the 5th choice. For option 6, system prints a message to obtain name of new category from admin. The new category will be set as `eventCat`. If input is not within 1 to 6, error message is printed and system loops back to `line-293`.

```
329     PRINT("Enter event name: ")
330     GET eventName
331     PRINT("Enter event organizer: ")
332     GET eventOrg
333     PRINT("Enter event start date in this format > [dd/mm/yyyy]: ")
334     GET eventDate
335     PRINT("Enter event start time in 24-hour format [eg: 16:30]: ")
336     GET eventTime
337     PRINT("Enter event duration [eg: 2 days]: ")
338     GET eventDuration
339     PRINT("Enter the venue of the event: ")
340     GET eventLocation
341
342     CAPITALIZE FIRST LETTER OF EACH WORD IN eventName
343     CAPITALIZE FIRST LETTER OF EACH WORD IN eventLocation
```

Now that the category has been chosen, system asks for event details such as `eventName`, `eventOrg`, `eventDate` and so on. For aesthetic purposes, the first letter of each word in `eventName` and `eventLocation` is capitalized.

```
345     WHILE True
346         PRINT("Is your event free-of-charge? [Y/N]: ")
347         GET eventFee
348         IF eventFee = "Y" THEN
349             eventPrice = "0.00"
350             BREAK
351         ELSE IF eventFee = "N" THEN
352             PRINT("Enter event price (per person) in RM [eg: 2.00]: ")
353             GET eventPrice
354             BREAK
355         ELSE
356             PRINT("Error: Please insert either [Y] or [N]")
357             CONTINUE
358         ENDIF
359     ENDWHILE
```

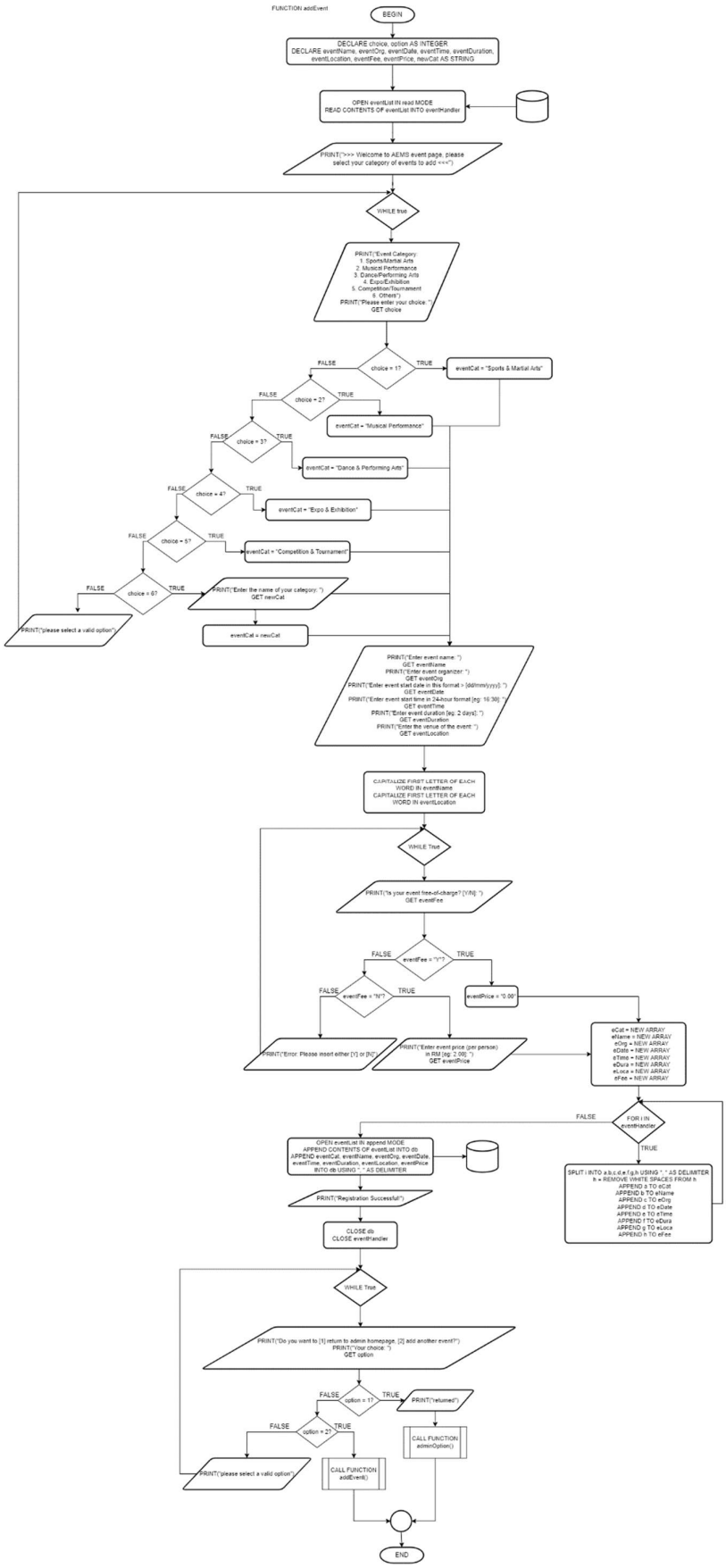
Next, system asks if the event is free-or-change. If it is free (`Y`) , `eventPrice` will be set to "`0.00`", otherwise (`N`) , admin may enter the event price. If neither Y or N is inserted, error message is printed and system loops back to `line-345`.

```
361     eCat = NEW ARRAY
362     eName = NEW ARRAY
363     eOrg = NEW ARRAY
364     eDate = NEW ARRAY
365     eTime = NEW ARRAY
366     eDura = NEW ARRAY
367     eLoca = NEW ARRAY
368     eFee = NEW ARRAY
369
370     FOR i IN eventHandler
371         SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
372         h = REMOVE WHITE SPACES FROM h
373         APPEND a TO eCat
374         APPEND b TO eName
375         APPEND c TO eOrg
376         APPEND d TO eDate
377         APPEND e TO eTime
378         APPEND f TO eDura
379         APPEND g TO eLoca
380         APPEND h TO eFee
381     ENDFOR
```

Multiple new arrays are created to append contents in. The strings in `eventHandler` are split into a, b, c, d, e, f, g, h and each are appended to the corresponding arrays. This process repeats until contents in `eventHandler` have been split and appended. Black spaces was also removed from string h.

```
383     OPEN eventList IN append MODE
384     APPEND CONTENTS OF eventList INTO db
385     APPEND eventCat, eventName, eventOrg, eventDate, eventTime, eventDuration, eventLocation, eventPrice INTO db USING ", " AS DELIMITER
386     PRINT("Registration Successful!")
387     CLOSE db
388     CLOSE eventHandler
389
390     WHILE True
391         PRINT("Do you want to [1] return to admin homepage, [2] add another event?")
392         PRINT("Your choice: ")
393         GET option
394         IF option = 1 THEN
395             PRINT("returned")
396             CALL FUNCTION adminOption()
397         ELSEIF option = 2 THEN
398             CALL FUNCTION addEvent()
399         ELSE
400             PRINT("please select a valid option")
401             CONTINUE
402         ENDIF
403     ENDWHILE
404 END
```

To add the event into system, `eventList` is opened and its contents are appended into `db`. The recently created new arrays are appended to the `db` with "`,  `" that acts as a separator among strings. Registration successful message is printed to notify admin that event has been added into system. System then asks whether the admin wants to return to admin homepage or add another event. If select 1, system calls `adminOption()`; if select 2, calls `addEvent()`; otherwise, system prints invalid option message and loops to `line-390` for user to re-enter option.


FUNCTION addEvent
BEGIN
DECLARE choice, option AS INTEGER
DECLARE eventName, eventOrg, eventDate, eventTime, eventDuration, eventLocation, eventFee, eventPrice, newCat AS STRING
OPEN eventList.IN read MODE
READ CONTENTS OF eventList INTO eventHandler
PRINT(">>> Welcome to AEMS event page, please select your category of events to add <<<")
WHILE true
PRINT("Event Category:
1. Sports/Martial Arts
2. Musical Performance
3. Dance/Performing Arts
4. Expo/Exhibition
5. Competition/Tournament
6. Others");
PRINT("Please enter your choice: ")
GET choice
choice = 1?
eventCat = "Sports & Martial Arts"
choice = 2?
eventCat = "Musical Performance"
choice = 3?
eventCat = "Dance & Performing Arts"
choice = 4?
eventCat = "Expo & Exhibition"
choice = 5?
eventCat = "Competition & Tournament"
choice = 6?
PRINT("Enter the name of your category: ")
GET newCat
eventCat = newCat
PRINT("please select a valid option")
PRINT("Enter event name: ")
GET eventName
PRINT("Enter event organizer: ")
GET eventOrg
PRINT("Enter event start date in this format > [dd/mm/yyyy]: ")
GET eventDate
PRINT("Enter event start time in 24-hour format [eg: 16:30] :")
GET eventTime
PRINT("Enter event duration [eg: 2 day(s): ")
GET eventDuration
PRINT("Enter the venue of the event: ")
GET eventLocation
CAPITALIZE FIRST LETTER OF EACH WORD IN eventName
CAPITALIZE FIRST LETTER OF EACH WORD IN eventLocation
WHILE True
PRINT("Is your event free-of-charge? (Y/N): ")
GET eventFee
eventFee = "Y"?
eventFee = "N"?
eventPrice = "0.00"
PRINT("Error. Please insert either [Y] or [N]")
PRINT("Enter event price (per person) in RM [eg: 2.00]: ")
GET eventPrice
eCat = NEW ARRAY
eName = NEW ARRAY
eOrg = NEW ARRAY
eDate = NEW ARRAY
eTime = NEW ARRAY
eDura = NEW ARRAY
eLoca = NEW ARRAY
eFee = NEW ARRAY
FOR i IN eventHandler
SPLIT i INTO a b c d e f g h USING ", " AS DELIMITER
h = REMOVE WHITE SPACES FROM h
APPEND a TO eCat
APPEND b TO eName
APPEND c TO eOrg
APPEND d TO eDate
APPEND e TO eTime
APPEND f TO eDura
APPEND g TO eLoca
APPEND h TO eFee
OPEN eventList.IN append MODE
APPEND CONTENTS OF eventList INTO db
APPEND eventCat, eventName, eventOrg, eventDate, eventTime, eventDuration, eventLocation, eventPrice INTO db USING ", " AS DELIMITER
PRINT("Registration Successful!")
CLOSE db
CLOSE eventHandler
WHILE True
PRINT("Do you want to [1] return to admin homepage, [2] add another event?")
PRINT("Your choice: ")
GET option
option = 1?
PRINT("returned")
CALL FUNCTION adminOption()
option = 2?
CALL FUNCTION addEvent()
PRINT("please select a valid option")
END

Selecting category produces multiple possible outcomes. The options are placed in decision boxes, each with two branches `TRUE` and `FALSE`. The `FALSE` branch of `choice = 6?` loops back to `WHILE True` for iteration, while all `TRUE` branches will go to the *input/output* box that obtains event details from the admin. After that, another decision regarding event price is made, similar flow occurs here where `TRUE` branches proceed to next step, `FALSE` branch loops. The splitting of `i` and appending of `a, b, c, d, e, f, g, h` processes iterate with an arrow going back to the loop `FOR I IN eventHandler`. When all contents are complete with split and append, `for loop` breaks and takes the `FALSE` branch. Next, `eventList` is appended into `db` and new event details are appended into `db`. Lastly, admin will be returned to `adminOption()` or `addEvent()` depending on the option selected is `1` or `2`. If neither 1 nor 2 is inserted, system takes the `FALSE` branch of `option = 2?` and loops back to `WHILE True`.

>>> modifyEvent() extension

```
FUNCTION modifyEvent

BEGIN
    DECLARE selection, counter, choice AS INTEGER
    DECLARE modCat, mod2, mod3, mod4, mod5, mod6, mod7, mod8 AS STRING
    OPEN eventList IN read MODE
    READ CONTENTS OF eventList INTO eventdb
    counter = 1
    PRINT("Which event would you like to amend?")
    FOR i IN eventdb
        SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
        PRINT(counter,"|", b)
        counter = counter + 1
    ENDFOR
    CLOSE eventdb

    PRINT("Your choice: ")
    GET selection
    selection = selection - 1

    OPEN eventList IN read MODE
    READ CONTENTS OF eventList INTO modifydb
    COPY CONTENTS FROM modifydb TO modList
    editList = NEW ARRAY
    newList = modList
    tempIndex = COPY CONTENT FROM modList AT ARRAY LOCATION selection
    CLOSE modifydb

    PRINT(">>> Current Event Details:")
    PRINT(tempIndex)
```

`modifyEvent()` allows admin to modify the existing event details. First, contents of `eventList` are read into `eventdb`. The system prints the event titles of all existing events using for loop and split just like `line-724` as explained in `memberLogin()`. Selection represents the user choice and `selection - 1` is required to return the correct event due to strings being stored at array locations starting from 0, which was also explained in `memberLogin()`. Contents in `eventList` is read into `modifydb`, which is then copied to the `modList`. The new array `editList` is created for later use; the contents of `modList` is copied to `newList`, also for later use. The contents located at array location `selection` from the `modList` is copied into `tempIndex`, and the system prints the `tempIndex` to display current event details

```
    WHILE True
        PRINT("Select New Event Category:
        1. Sports/Martial Arts
        2. Musical Performance
        3. Dance/Performing Arts
        4. Expo/Exhibition
        5. Competition/Tournament
        6. Others")
        PRINT("Please enter your choice: ")
        GET choice
        IF choice = 1 THEN
            mod1 = "Sports & Martial Arts"
            BREAK
        ELSE IF choice = 2 THEN
            mod1 = "Musical Performance"
            BREAK
        ELSE IF choice = 3 THEN
            mod1 = "Dance & Performing Arts"
            BREAK
        ELSE IF choice = 4 THEN
            mod1 = "Expo & Exhibition"
            BREAK
        ELSE IF choice = 5 THEN
            mod1 = "Competition & Tournament"
            BREAK
        ELSE IF choice = 6 THEN
            PRINT("Enter the name of your category: ")
            GET modCat
            mod1 = modCat
            BREAK
        ENDIF
    ENDWHILE
```

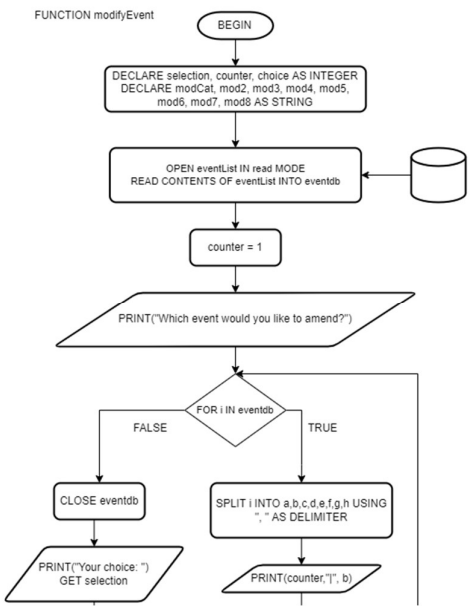
After the system retrieved current event details, the admin needs to reinsert all new event details, starting from event category. All existing categories are printed, from choice `1` to `5`, each represents a category. Choice `6` allows admin to create a new category. The input is saved as `modCat` variable. The outcome of each choice is saved as `mod1`.

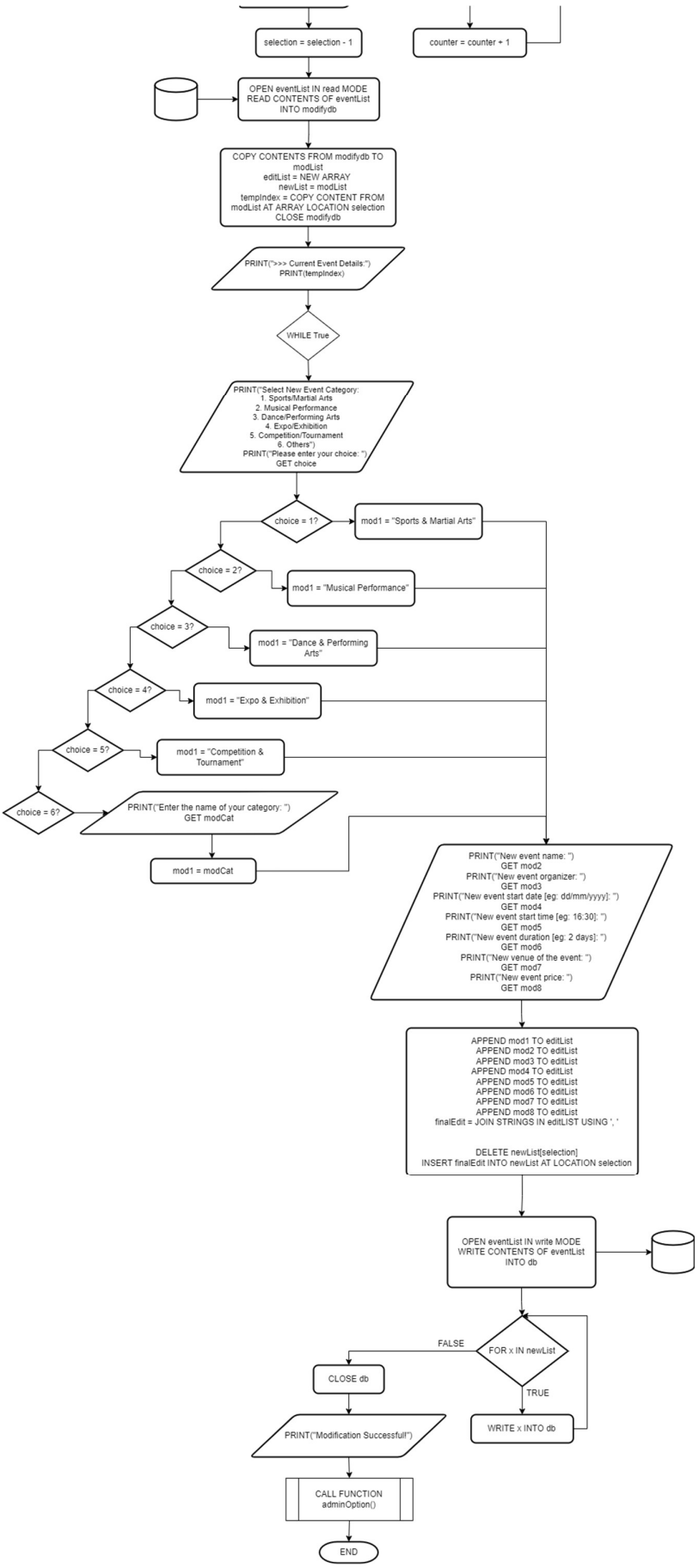
```
    PRINT("New event name: ")
    GET mod2
    PRINT("New event organizer: ")
    GET mod3
    PRINT("New event start date [eg: dd/mm/yyyy]: ")
    GET mod4
    PRINT("New event start time [eg: 16:30]: ")
    GET mod5
    PRINT("New event duration [eg: 2 days]: ")
    GET mod6
    PRINT("New venue of the event: ")
    GET mod7
    PRINT("New event price: ")
    GET mod8
```

The following event details have to be inserted manually. Input for different event details are saved as different variable names starting from `mod2` to `mod8`.

```
574     APPEND mod1 TO editList
575     APPEND mod2 TO editList
576     APPEND mod3 TO editList
577     APPEND mod4 TO editList
578     APPEND mod5 TO editList
579     APPEND mod6 TO editList
580     APPEND mod7 TO editList
581     APPEND mod8 TO editList
582     finalEdit = JOIN STRINGS IN editLIST USING ', '
583
584     DELETE newList[selection]
585     INSERT finalEdit INTO newList AT LOCATION selection
586
587     OPEN eventList IN write MODE
588     WRITE CONTENTS OF eventList INTO db
589     FOR x IN newList
590         WRITE x INTO db
591     ENDFOR
592     CLOSE db
593     PRINT("Modification Successful!")
594     CALL FUNCTION adminOption()
595 END
```

Now the system has received all new event details, the variables are appended to the `editList` created before. All the event details are joined as a single string using the '`, `' and is saved as `finalEdit`. The old contents at array location selection in the `newList` is deleted, replaced by writing `finalEdit` into the same location. To edit the `eventList`, which is the permanent list that stores event details, AEGIS first writes the current contents of `evenList` into `db`. New contents (`x`) from `newList` is written into the `db`. Once all new event details are written into `db`, the modify event procedure completes. System prints a modification successful message and redirect admins to the `adminOption()` page.



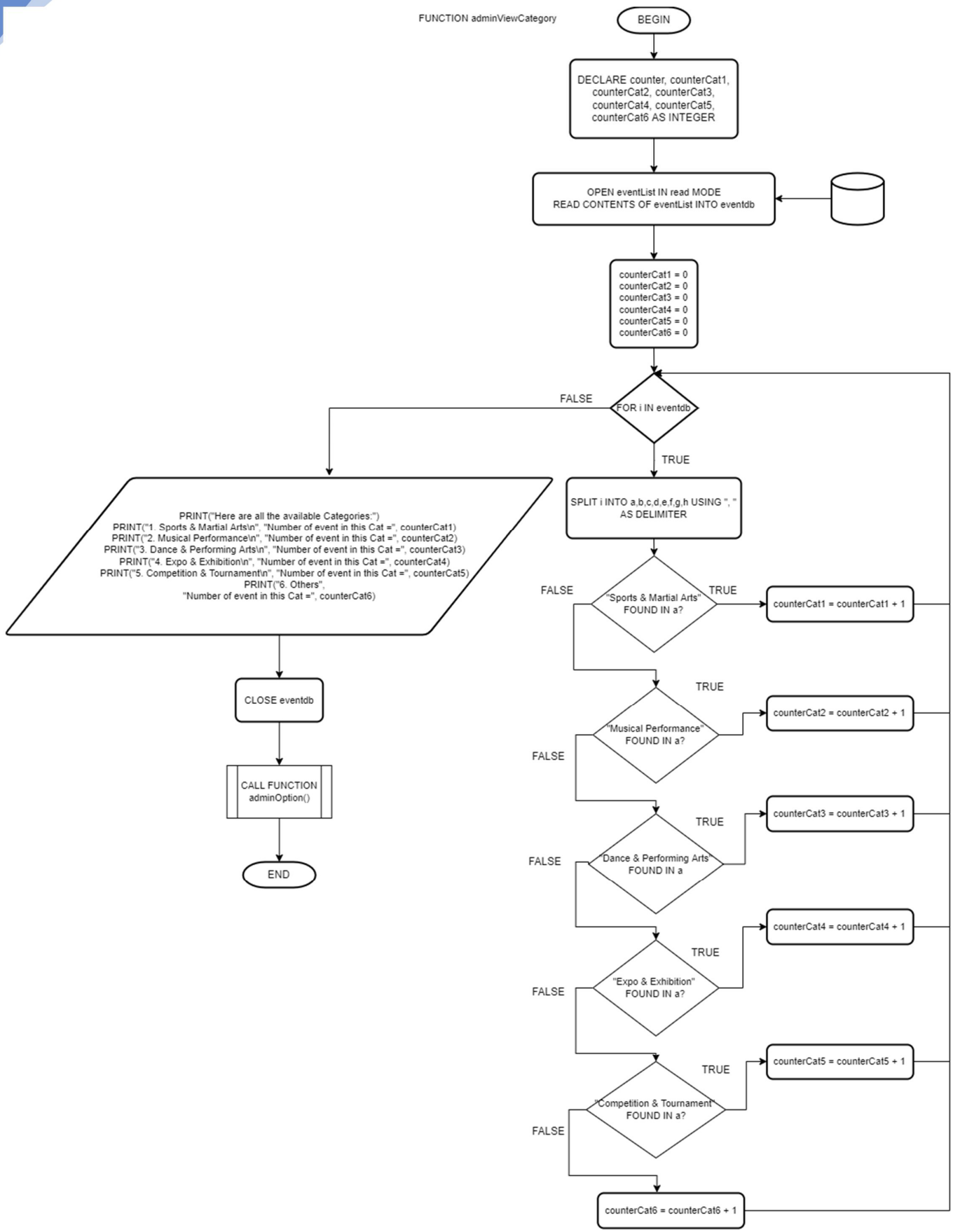

selection = selection - 1
counter = counter + 1
OPEN eventList IN read MODE
READ CONTENTS OF eventList
INTO modifydb
COPY CONTENTS FROM modifydb TO modList
editList = NEW ARRAY
newList = modList
tempIndex = COPY CONTENT FROM modList AT ARRAY LOCATION selection
CLOSE modifydb
PRINT(">>> Current Event Details:")
PRINT(tempIndex)
WHILE True
PRINT("Select New Event Category:
1. Sports/Martial Arts
2. Musical Performance
3. Dance/Performing Arts
4. Expo/Exhibition
5. Competition/Tournament
6. Others")
PRINT("Please enter your choice: ")
GET choice
choice = 1?
mod1 = "Sports & Martial Arts"
choice = 2?
mod1 = "Musical Performance"
choice = 3?
mod1 = "Dance & Performing Arts"
choice = 4?
mod1 = "Expo & Exhibition"
choice = 5?
mod1 = "Competition & Tournament"
choice = 6?
PRINT("Enter the name of your category: ")
GET modCat
mod1 = modCat
PRINT("New event name: ")
GET mod2
PRINT("New event organizer: ")
GET mod3
PRINT("New event start date [eg: dd/mm/yyyy]: ")
GET mod4
PRINT("New event start time [eg: 16:30]: ")
GET mod5
PRINT("New event duration [eg: 2 days]: ")
GET mod6
PRINT("New venue of the event: ")
GET mod7
PRINT("New event price: ")
GET mod8
APPEND mod1 TO editList
APPEND mod2 TO editList
APPEND mod3 TO editList
APPEND mod4 TO editList
APPEND mod5 TO editList
APPEND mod6 TO editList
APPEND mod7 TO editList
APPEND mod8 TO editList
finalEdit = JOIN STRINGS IN editLIST USING ', '
DELETE newList[selection]
INSERT finalEdit INTO newList AT LOCATION selection
OPEN eventList IN write MODE
WRITE CONTENTS OF eventList
INTO db
FOR x IN newList
FALSE
TRUE
CLOSE db
WRITE x INTO db
PRINT("Modification Successfull")
CALL FUNCTION
adminOption()
END

display options:

```
>>> adminViewCategory()
```

```
FUNCTION adminViewCategory

BEGIN
    DECLARE counter, counterCat1, counterCat2, counterCat3, counterCat4, counterCat5, counterCat6 AS INTEGER
    OPEN eventList IN read MODE
    READ CONTENTS OF eventList INTO eventdb
    counterCat1 = 0
    counterCat2 = 0
    counterCat3 = 0
    counterCat4 = 0
    counterCat5 = 0
    counterCat6 = 0
    FOR i IN eventdb
        SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
        IF "Sports & Martial Arts" FOUND IN a THEN
            counterCat1 = counterCat1 + 1
        ELSE IF "Musical Performance" FOUND IN a THEN
            counterCat2 = counterCat2 + 1
        ELSE IF "Dance & Performing Arts" FOUND IN a THEN
            counterCat3 = counterCat3 + 1
        ELSE IF "Expo & Exhibition" FOUND IN a THEN
            counterCat4 = counterCat4 + 1
        ELSE IF "Competition & Tournament" FOUND IN a THEN
            counterCat5 = counterCat5 + 1
        ELSE
            counterCat6 = counterCat6 + 1
        ENDIF
    ENDFOR

    PRINT("Here are all the available Categories:")
    PRINT("1. Sports & Martial Arts\n", "Number of event in this Cat =", counterCat1)
    PRINT("2. Musical Performance\n", "Number of event in this Cat =", counterCat2)
    PRINT("3. Dance & Performing Arts\n", "Number of event in this Cat =", counterCat3)
    PRINT("4. Expo & Exhibition\n", "Number of event in this Cat =", counterCat4)
    PRINT("5. Competition & Tournament\n", "Number of event in this Cat =", counterCat5)
    PRINT("6. Others",
    "Number of event in this Cat =", counterCat6)
    CLOSE eventdb
    CALL FUNCTION adminOption()
END
```

The `counterCat1` to `counterCat6` are used by the respective event categories. First of all, as usual, we read `eventList` into `eventdb`. In a for loop, contents in `eventdb` which are event details, are split into smaller strings a, b, c, d, e, f, g, h. Event category is located at `a`, therefore the `counterCat` increases by 1 only when `a` is found in `eventdb`. The `counterCat` represents the number of events in the system, therefore they are set as 0 if no events are found. Once all contents in have been accessed, AEGIS prints the total number of events in each category. Lastly, user is redirected to `adminOption()`.

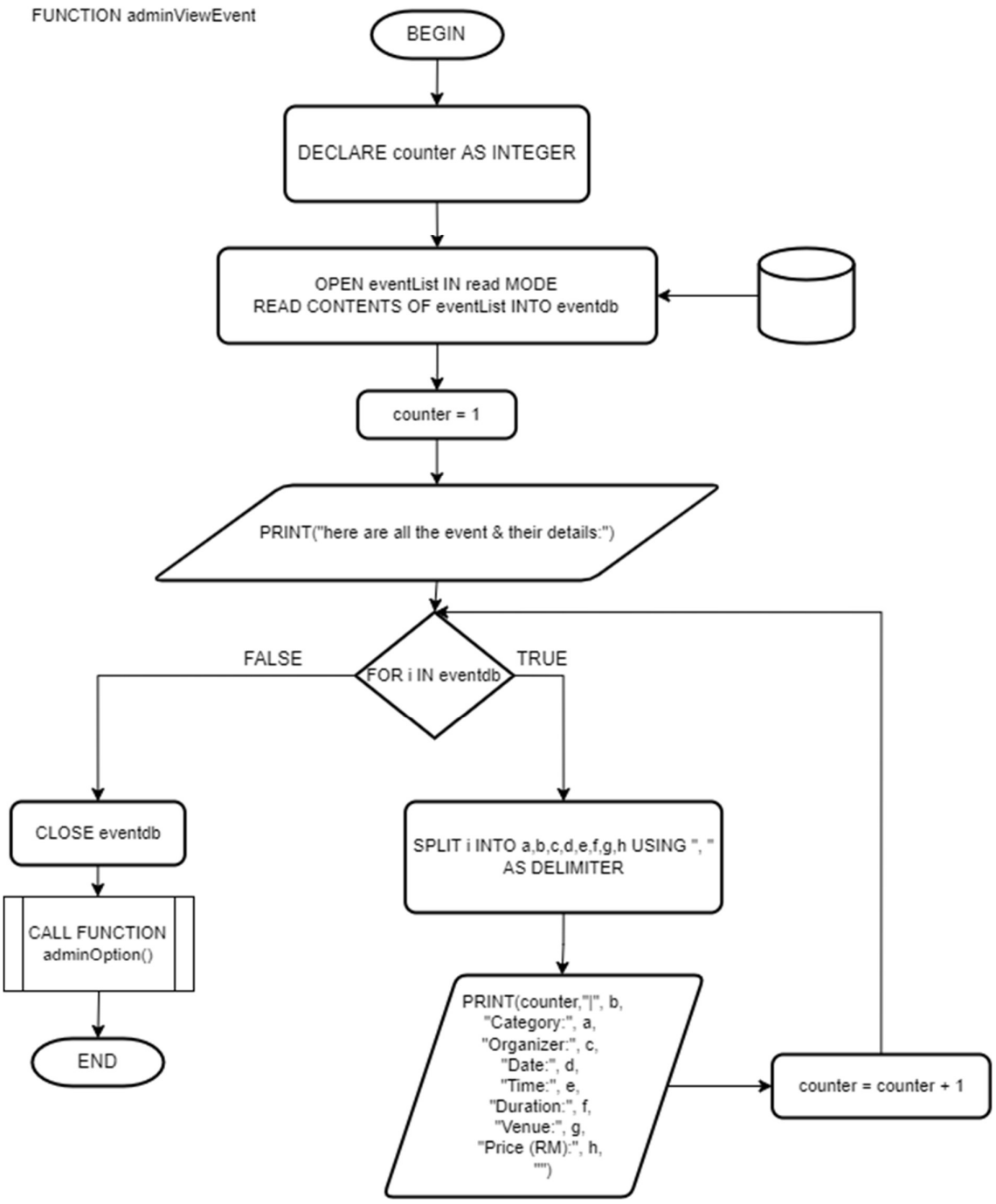

FUNCTION adminViewCategory
BEGIN
DECLARE counter, counterCat1, counterCat2, counterCat3, counterCat4, counterCat5, counterCat6 AS INTEGER
OPEN eventList IN read MODE
READ CONTENTS OF eventList INTO eventdb
counterCat1 = 0
counterCat2 = 0
counterCat3 = 0
counterCat4 = 0
counterCat5 = 0
counterCat6 = 0
FOR i IN eventdb
FALSE
TRUE
SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
"Sports & Martial Arts" FOUND IN a?
counterCat1 = counterCat1 + 1
"Musical Performance" FOUND IN a?
counterCat2 = counterCat2 + 1
"Dance & Performing Arts" FOUND IN a
counterCat3 = counterCat3 + 1
"Expo & Exhibition" FOUND IN a?
counterCat4 = counterCat4 + 1
"Competition & Tournament" FOUND IN a?
counterCat5 = counterCat5 + 1
counterCat6 = counterCat6 + 1
PRINT("Here are all the available Categories:")
PRINT("1. Sports & Martial Arts\n", "Number of event in this Cat =", counterCat1)
PRINT("2. Musical Performance\n", "Number of event in this Cat =", counterCat2)
PRINT("3. Dance & Performing Arts\n", "Number of event in this Cat =", counterCat3)
PRINT("4. Expo & Exhibition\n", "Number of event in this Cat =", counterCat4)
PRINT("5. Competition & Tournament\n", "Number of event in this Cat =", counterCat5)
PRINT("6. Others",
"Number of event in this Cat =", counterCat6)
CLOSE eventdb
CALL FUNCTION adminOption()
END

After reading `eventList` into `evendb`, all the `counterCat` are defined as `0`. `For loop` begins by splitting contents into small units and retrieve `a`. For each event category, if it is found in a, it takes the `TRUE` branch and the corresponding `counterCat` increase by `1`. If the category is not found in a , it takes the `FALSE` branch and goes to next *decision* box. All the outputs loop back to the `for loop` and goes on for a few more rounds until all contents are accessed. The `for loop` ends and prints number of events available in each category. Function `adminOption()` is called at the end.

```
>>> adminViewEvent()
```

```
449  FUNCTION adminViewEvent
450
451  BEGIN
452      DECLARE counter AS INTEGER
453      OPEN eventList IN read MODE
454      READ CONTENTS OF eventList INTO eventdb
455      counter = 1
456      PRINT("here are all the event & their details:")
457      FOR i IN eventdb
458          SPLIT i INTO a,b,c,d,e,f,g,h USING ", " AS DELIMITER
459          PRINT(counter,"|", b,
460          "Category:", a,
461          "Organizer:", c,
462          "Date:", d,
463          "Time:", e,
464          "Duration:", f,
465          "Venue:", g,
466          "Price (RM):", h,
467          "")
468          counter = counter + 1
469      ENDFOR
470      CLOSE eventdb
471      CALL FUNCTION adminOption()
472  END
```

`adminViewEvent()` is allows admins to view all events and details. Similar to previously explained functions, `eventList` is opened and read, contents in `eventdb` are split into individual strings for each event detail and printed. The function ends with calling `adminOption()`.

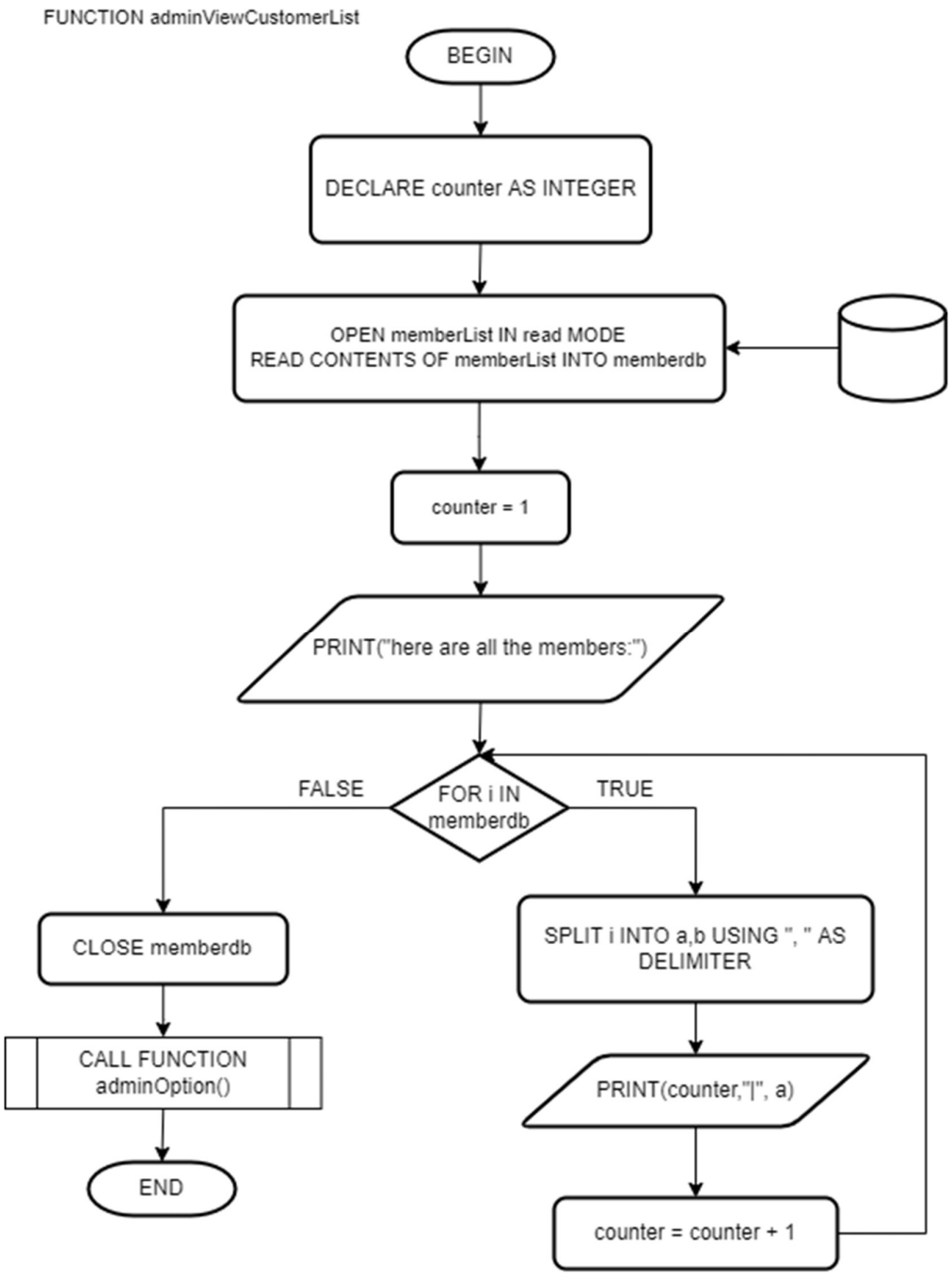


Upon opening and reading `eventList`, contents are split and printed in a for loop until all events have been printed. Once completed, `eventdb` is closed and `adminOption()` is called.

```
>>> adminViewCustomerList()
```

```
FUNCTION adminViewCustomerList

BEGIN
    DECLARE counter AS INTEGER
    OPEN memberList IN read MODE
    READ CONTENTS OF memberList INTO memberdb
    counter = 1
    PRINT("here are all the members:")
    FOR i IN memberdb
        SPLIT i INTO a,b USING ", " AS DELIMITER
        PRINT(counter,"|", a)
        counter = counter + 1
    ENDFOR
    CLOSE memberdb
    CALL FUNCTION adminOption()
END
```

`adminViewCustomerList()` allows admins to view all registered customers. Firstly, `memberList` which is the permanent list for members, is opened and read into `memberdb`. The original string contains member username and member password; therefore, the string is split into two, with member username as `a` and member password as `b`. `a` is printed along with the `counter` to indicate the number of customers. After first customer is printed, `counter` increases by `1` and loops. The process iterates until all members are printed. Lastly `memberdb` is closed and `adminOption()` is called.

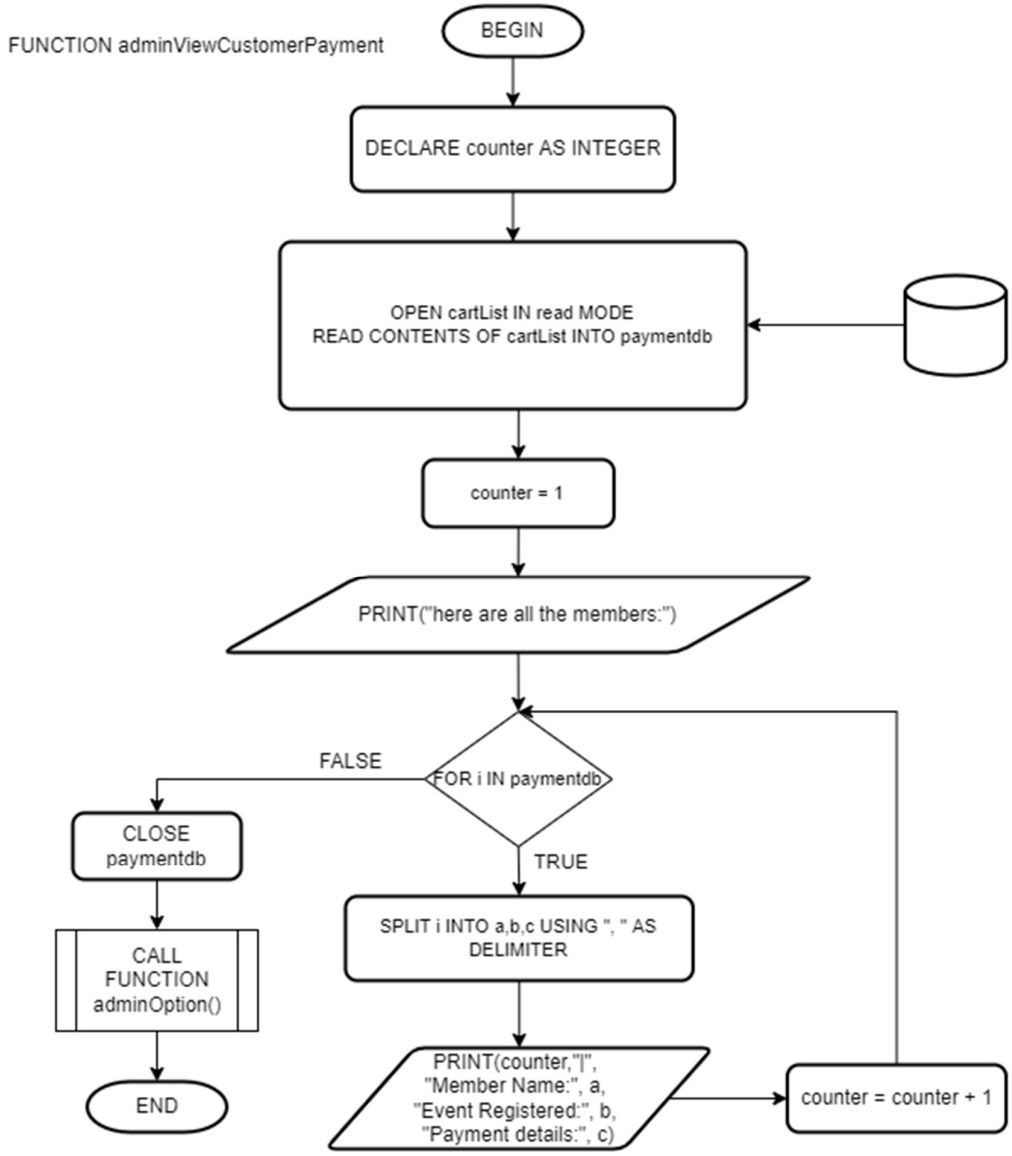


After opening and reading `memberList`, contents or `memberdb` are split into `a` and `b`. After printing `a`, `1` is added to counter and the system loops until all members have been printed. Lastly function `adminOption()` is called.

```
>>> adminViewCustomerPayment()
```

```
874  FUNCTION adminViewCustomerPayment
875
876  BEGIN
877      DECLARE counter AS INTEGER
878      OPEN cartList IN read MODE
879      READ CONTENTS OF cartList INTO paymentdb
880      counter = 1
881      PRINT("here are all the members:")
882      FOR i IN paymentdb
883          SPLIT i INTO a,b,c USING ", " AS DELIMITER
884          PRINT(counter,"|",
885          "Member Name:", a,
886          "Event Registered:", b,
887          "Payment details:", c)
888          counter = counter + 1
889      ENDFOR
890      CLOSE paymentdb
891      CALL FUNCTION adminOption()
892  END
```

`adminViewCustomerPayment()` applies the same set of rules as `adminViewCustomerList()`. The function begins with opening `cartList` and reading the contents into `paymentdb`. The content in `paymentdb` is split into 3 strings, a, b, and c with the "`, `" as separator. `a` is printed as member name, `b` is printed as event registered while `c` is payment details. `1` is added to counter and the process loops. Finally, `paymendb` is closed and system redirected to `adminOption()`.



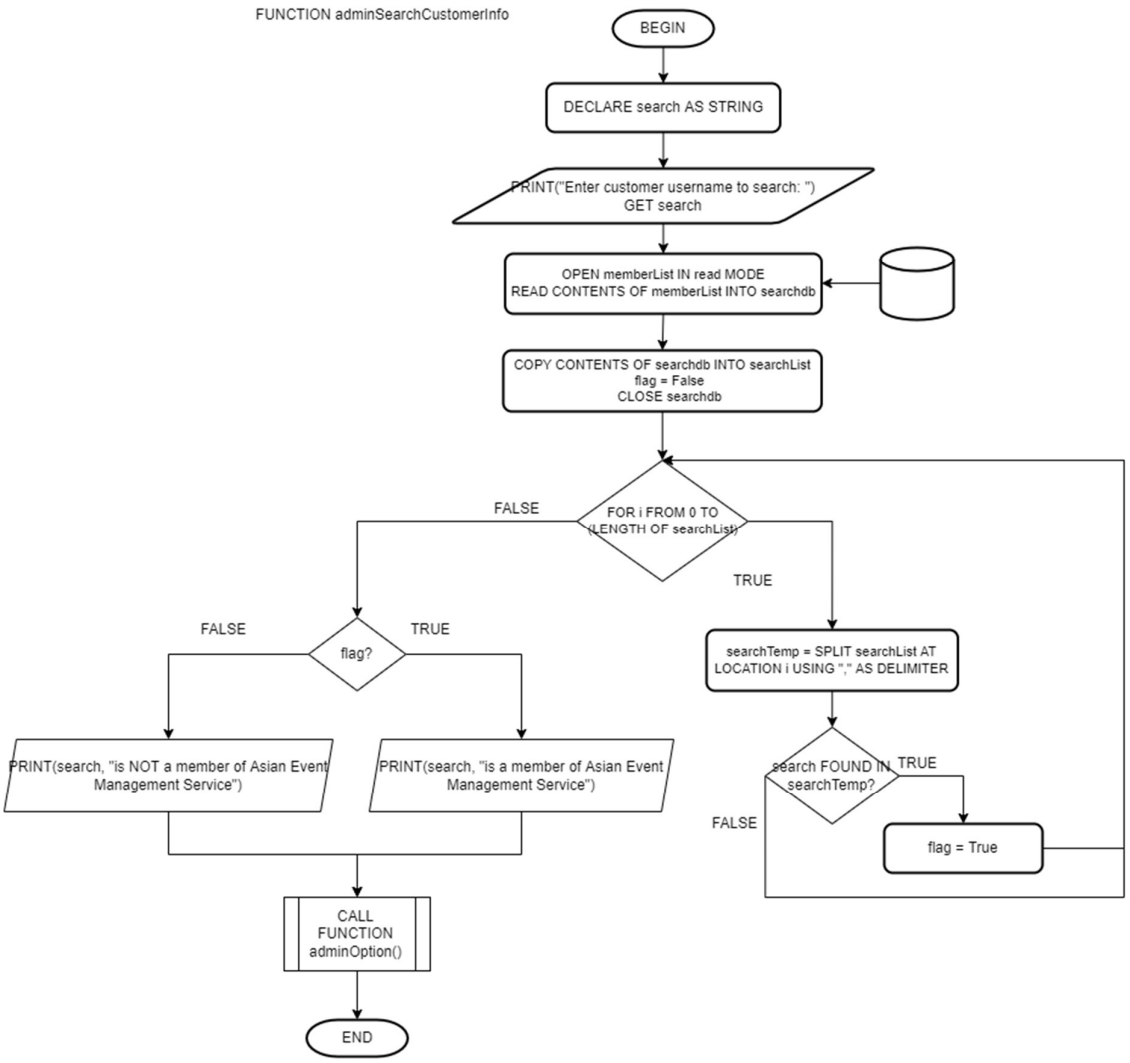
The flow of `adminViewCustomerPayment()` is same as `adminViewCustomerList()`.

search options:

```
>>> adminSearchCustomerInfo()
```

```
895  FUNCTION adminSearchCustomerInfo
896
897  BEGIN
898      DECLARE search AS STRING
899      PRINT("Enter customer username to search: ")
900      GET search
901      OPEN memberList IN read MODE
902      READ CONTENTS OF memberList INTO searchdb
903      COPY CONTENTS OF searchdb INTO searchList
904      flag = False
905      CLOSE searchdb
906      FOR i FROM 0 TO (LENGTH OF searchList)
907          searchTemp = SPLIT searchList AT LOCATION i USING "," AS DELIMITER
908          IF search FOUND IN searchTemp THEN
909              flag = True
910          ELSE
911              PASS
912          ENDIF
913      ENDFOR
914
915      IF flag THEN
916          PRINT(search, "is a member of Asian Event Management Service")
917      ELSE
918          PRINT(search, "is NOT a member of Asian Event Management Service")
919      ENDIF
920      CALL FUNCTION adminOption()
921  END
```

`adminSearchCustomerInfo()` is used for admins to search registered customers. AEGIS first asks the admin to insert the customer username. Then, the system opens the `memberList` and read the contents into `searchdb`. The contents in `searchdb` is then copied to `searchList`. For contents within the range of `0` to `length of searchList`, the `searchList` at location `i` is split using "`,`" and saved as `searchTemp`. If the input given by admin is found in `searchTemp`, flag is `True`, and the system loops back to top. If user input is not found the system also loop back to `line-906` to check the remaining contents. If `flag` is `True`, the customer is a member of AEMS, otherwise, it means the user has not register as a member. The function ends and redirects to `adminOption()`.



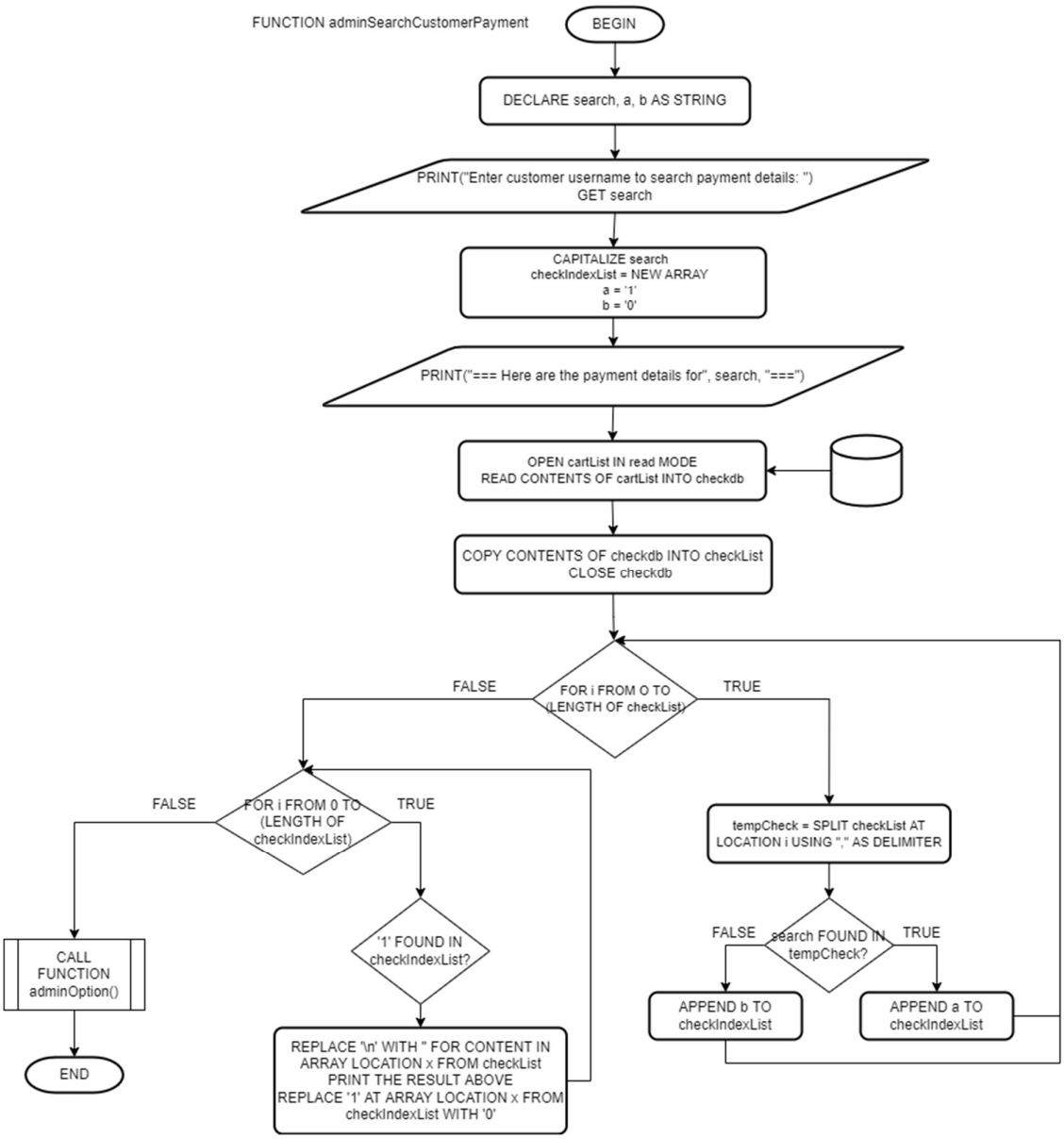
The system starts by getting user input, then open and read the `memberList`. Contents of `searchdb` is copied into a `searchList`. All the contents (`i`) in the searchList, will be split at location `i` and saved as `searchTemp`. If the previous user input is found in `searchTemp`, then `flag` is `True` and loops, otherwise the system also loops. If `flag` is `TRUE`, system prints a message indicating the user is member of AEMS while if `flag` is `FALSE`, system indicates user is not a member. `adminOption()` is called at the end.

```
>>> adminSearchCustomerPayment()
```

```
FUNCTION adminSearchCustomerPayment

BEGIN
    DECLARE search, a, b AS STRING
    PRINT("Enter customer username to search payment details: ")
    GET search
    CAPITALIZE search
    checkIndexList = NEW ARRAY
    a = '1'
    b = '0'
    PRINT("=== Here are the payment details for", search, "===")
    OPEN cartList IN read MODE
    READ CONTENTS OF cartList INTO checkdb
    COPY CONTENTS OF checkdb INTO checkList
    CLOSE checkdb
    FOR i FROM 0 TO (LENGTH OF checkList)
        tempCheck = SPLIT checkList AT LOCATION i USING "," AS DELIMITER
        IF search FOUND IN tempCheck THEN
            APPEND a TO checkIndexList
        ELSE
            APPEND b TO checkIndexList
        ENDIF
    ENDFOR

    FOR i FROM 0 TO (LENGTH OF checkIndexList)
        IF '1' FOUND IN checkIndexList THEN
            x = ARRAY LOCATION OF checkIndexList WHERE '1' IS PRESENT
            REPLACE '\n' WITH '' FOR CONTENT IN ARRAY LOCATION x FROM checkList
            PRINT THE RESULT ABOVE
            REPLACE '1' AT ARRAY LOCATION x FROM checkIndexList WITH '0'
        ELSE
            PASS
        ENDIF
    ENDFOR
    CALL FUNCTION adminOption()
END
```

`adminSearchCustomerPayment()` is used by admins to check the payment details of customers. First username of customer to be checked needs to be inputted. `checkIndexList` is created as a new array, and a is defines as `'1'`, b is defined as `'0'`. The `cartList` is opened and read into `checkdb`, and the contents is then copied to `checkList`. The for loop begins, in each loop, the `checkList` will be split at location `i` and saved as `tempCheck`. When the customer username is found in `tempCheck`, `a` is appended to `checkIndexList`; otherwise, `b` is appended to `checkIndexList`. When all contents in `checkList` have gone through the loop, the loop ends. Another for loop is initiated to check the `checkIndexList`. Similar to the "*view purchased event*" section under `memberLogin()`, if `'1'` is found in the `checkIndexList`, `x` is defined as the array location where `'1'` exists. `\n` is removed by replaced it with empty string `''` at array location `x` in the `checkList`. The result will be printed and to once printed, we do not want to print it again to avoid duplication. Therefore `'1'` is replaced with `'0'` at array location `x` in the

`checkIndexList`. The process iterates until all strings have been checked. Once completed, user is redirected to `adminOption()`.



After getting user input (variable `search`), new array `checkIndexList` is created. cartList is opened and read, and then contents is copied into checkdb. For loop begins to check if `search` present in `tempCheck`. If present, `a` is appended to `checkIndexList`, otherwise append `b`. Both outputs go back to the loop for checking of remaining contents. Once all contents checked, system follows the `FALSE` branch from the loop and goes to the second `for loop`. If `'1'` is found in `checkIndexList`, multiple process is carried out as explained in above pseudocode. Finally, then function ends with calling `adminOption()`.

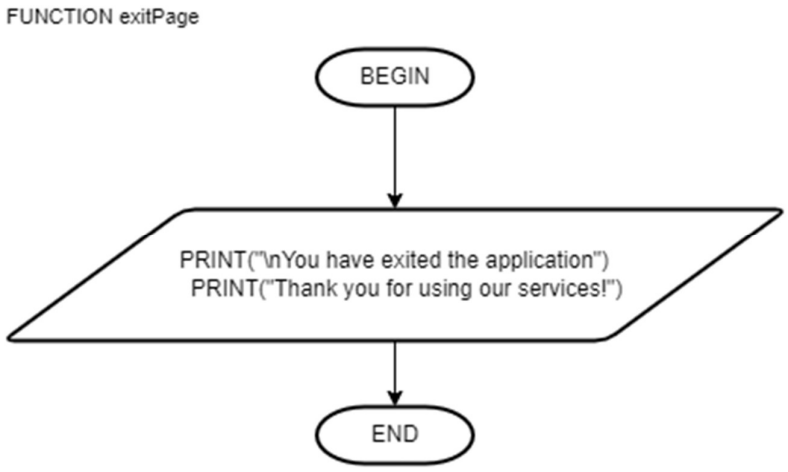
> exit page design

Exit page is designed for users to exit the AEGIS program.

```
>>> exitPage()
```

```
31  FUNCTION exitPage
32
33  BEGIN
34      PRINT("\nYou have exited the application")
35      PRINT("Thank you for using our services!")
36  END
```

Once `exitPage()` function is called, the system notifies the user he or she has exited the program and prints a thank you message.



This is the flowchart for `exitPage()`. The function ends after printing two messages.

Program Source Code with Explanation

> import datetime

```
8
9   import datetime # import datetme from python library for later use
10
```

Building on from the pseudocodes, we know that AEGIS will eventually utilize the `today()` function from the `datetime` module in the payment system. As such, it is a good practice to declare it at the start of the code, so that (1) programmers can see all the dependencies of the program, and (2) avoid situations where the programmer is trying to use a module before it is imported from library.

> def homepage()

```
12  def homepage():
13      print("=== Welcome to Asian Event Management Service (AEMS) ===")
14      print("Please select your position:")
15      print("1. Admin")
16      print("2. Customer")
17      print("3. Exit")
18      option = int(input("Please select an option: "))
19      if option == 1:
20          adminHome()
21      elif option == 2:
22          guestOption()
23      elif option == 3:
24          exitPage()
25      else:
26          print("Please select a valid option")
27          homepage()
28
```

The `homepage()` function is defined at `line_12`, which contains a menu entailing 3 available options upon launching the application. Users can input an `integer` value at `line_18` which will be saved under the variable named `option`. This variable will then be assessed in an `if-else` statement which will branch out into other functions in accordance with the integer value chosen by the user. If user mistakenly inputted an invalid value, or value that exceeds the given range, the system will inform users to “Please select a valid option” and call the homepage() function for users to retry.

> def guestOption()

```
30  def guestOption():
31      print("\n>>> What would you like to do? \n")
32      print("1. View Category")
33      print("2. View Event")
34      print("3. Log in as a Member")
35      print("4. Register as a Member")
36      print("5. Exit")
37      choice = int(input("Your choice: "))
38      if choice == 1:
39          guestViewCategory()
40      elif choice == 2:
41          guestViewEvent()
42      elif choice == 3:
43          memberLogin()
44      elif choice == 4:
45          memberSignup()
46      elif choice == 5:
47          exitPage()
48      else:
49          print("option not supported, please try again")
50          guestOption()
51
```

Next, the `guestOption()` function is defined at `line_30` which contains 5 available options for a non-registered customer. Customer can choose to `view category`, `view event`, `login`, `register` or `exit the application` by inserting the appropriate integer value prompted by the terminal. Upon entry, the `if-else` statement at `line_38` will examine the value, and branch to the specific function as demanded by the user. However, if user were to enter options that are not in the supported categories, the system will print “option not supported, please try again” and re-call the `guestOption()` function for retry purposes.

> def adminHome()

```
53  def adminHome():
54      print("=== Please select an option below: ===")
55      print("1. Admin Login")
56      print("2. Admin Registration")
57      print("3. Go Back")
58      option = int(input("Your choice: "))
59      if option == 1:
60          adminLogin()
61      elif option == 2:
62          adminAuth()
63      elif option == 3:
64          homepage()
65      else:
66          print("Please select a valid option")
67          adminHome()
68
```

Similarly, the `adminHome()` function is defined at `line_53` which features a homepage interface designed for admins. Users can choose to `login as admin`, `register as admin`, or return to default homepage by calling the aforementioned `homepage()` function. If users mistakenly inserted an anomaly value outside of the range (1-3), the system will print the “Please select a valid option” and re-call the `adminHome()` function for users to re-attempt their entries.

> def adminAuth()

```
70  def adminAuth():
71      code = input("Please enter company referral code: ")
72      if code != "AEMS0001":
73          print("Incorrect referral code.")
74          print("do you want to (1) Try Again, or (2) Go Back ?")
75          choice = int(input("Your choice: "))
76          if choice == 1:
77              adminAuth()
78          elif choice == 2:
79              adminHome()
80          else:
81              print("option not supported, returning to homepage...")
82              homepage()
83      else:
84          print("Verification successful!")
85          adminSignup()
86
```

Should a user choose to register as admin, the `adminAuth()` function will be called, which is defined at `line_70`. This authentication method is important as we do not want unauthorized users to register as admin and take advantage of admin features. As such, a company referral is required before sign-up. The `if-else` statement at `line_72` contains the '`!=`' arithmetic operation, which means <if the referral code inserted is not the same as '`AEMS0001`'>, it will not continue. The user will be given the options to retry or go back to homepage if they fail to produce the correct referral code. However, if the code 'AEMS0001' is entered correctly, the verification process will be completed, and the system will then congratulate the user by proceeding with the admin sign-up process.

> def adminSignup()

```
88  def adminSignup():
89      adminHandler = open("adminList.txt", "r")
90      print(">>> Welcome to AEMS admin registration page, please follow the instruction below <<<")
91      username = input("Enter a username: ")
92      password1 = input("Enter a password: ")
93      password2 = input("Confirm your password: ")
94
95      # database
96      adminUN = []
97      adminPW = []
98      for i in adminHandler:
99          a,b = i.split(", ")
100         b = b.strip()
101         adminUN.append(a)
102         adminPW.append(b)
103
104     if password1 != password2:
105         print("Password does not match, please try again")
106         adminSignup()
107     elif len(password1) <= 8:
108         print("Password must be longer than 8 characters, please try again")
109         adminSignup()
110     elif username in adminUN:
111         print("Username already taken, please try again")
112         adminSignup()
113     else:
114         db = open("adminList.txt", "a")
115         db.write(username+", "+password1+"\n")
116         print("Registration Successful!")
117         db.close()
118     adminHandler.close()
119     adminHome()
120
```

The `adminSignup()` function, as declared in `line_88` is called when the verification process is successful in `adminAuth()`. This function will open `adminList.txt` in `read` mode under the variable `adminHandler` at the start and proceeds to obtain username and two passwords from the user under the variable `username`, `password1`, and `password2`.

After that, two empty list is created under the name `adminUN` and `adminPW` to store values in `adminList.txt` later on. Using a `for` loop, the system will read through each item in `adminList.txt` and split them up into two variables: `a` and `b` using the `split()` function with parameters of `", "` as delimiter. Any white/blank spaces on variable `b` will also be removed using the .strip() function. Then, the `append()` function is used to append all usernames found in `adminList.txt` into the `adminUN` list, and all respective passwords into the `adminPW` list.

Once that is completed, the error-checking phase begins at `line_104` which checks for 3 potential error scenarios: (1) password does not match, (2) password length is too short 8, and (3) username is already taken – using an `if-else` statement. When users have successfully passed the checker with no issues, `adminList.txt` will be opened in `append` mode under the variable `db`, and their associating username and password will be appended into the database using the `write()` function. Finally, the file handlers: `adminHandler` and `db` will be closed using the `close()` function, and users will be brought back to the admin homepage by called the `adminHome()` function.

> def adminLogin()

```
122 def adminLogin():
123     tempList = []
124     adminHandler = open("adminList.txt", "r")
125     while True:
126         print(">>> Admin Login Page <<<")
127         adminUN = input("Enter your username: ")
128         adminPW = input("Enter your password: ")
129         for i in adminHandler:
130             tempList.append(i)
131         if adminUN + ", " + adminPW + "\n" in tempList:
132             print("Login successful!")
133             print("Welcome back,", adminUN.capitalize())
134             adminOption()
135         else:
136             print("\n >>> invalid credentials... \n")
137             option = input("would you like to sign-up? \n [Y] for admin registration \n [N] to retry login \n Your choice: " )
138             if option == "N":
139                 continue #repeat while loop
140             elif option == "Y":
141                 adminAuth()
142             else:
143                 print("invalid option, returning to homepage...")
144                 homepage()
145         break
146     adminHandler.close()
147
```

Once the registration is completed, users can choose to login as admin using the `adminLogin()` function as declared in `line_122`. In this function, an empty, temporary list called `tempList` is created at the start. Then, `adminHandler` is used as a file handler to open `adminList.txt` in `read` mode to enable the remaining codes in this function to access the database. A `while` loop is created with the `Boolean` value `True` to enable users to retry if the credentials entered are invalid using the `continue` clause later on.

At `line_127` & `line_128`, an input line is displayed for the user to enter their usernames (saved as `adminUN`) and passwords (saved as `adminPW`), then contents from `adminList.txt` is copied into the `tempList` list. After that, an `if-else` statement is used to match the credentials inserted in `adminUN` and `adminPW` with the credentials present in the `tempList` by concatenating the strings using arithmetic operation '`+`'.

If the username and password combination match any strings found inside `tempList`, the login will be successful, and the user will be greeted with the first letter of their names capitalized using the `capitalize()` function, and they'll proceed to the next stage, `adminOption()`.

On the other hand, if the username and password combination does not match any strings found in `tempList`, the user will be prompt to either (1) retry with the `continue` clause, or (2) sign-up as admin using `adminAuth()`. Finally, the system will exit the `while` loop using the `break` clause and close the adminHandler using the `close()` function.

> def adminOption()

```
149 def adminOption():
150     print("\n What would you like to do? \n")
151     print("1. Add event")
152     print("2. Modify event records")
153     print("3. Display record details")
154     print("4. Search specific record")
155     print("5. Exit")
156     choice = int(input("Your choice: "))
157     if choice == 1:
158         addEvent()
159     elif choice == 2:
160         modifyEvent()
161     elif choice == 3:
162         print("display options: \n a. Event Category \n b. All Events \n c. Registered Customers \n d. Customer Payment")
163         selection = str(input("Enter your option: "))
164         if selection == "a":
165             adminViewCategory()
166         elif selection == "b":
167             adminViewEvent()
168         elif selection == "c":
169             adminViewCustomerList()
170         elif selection == "d":
171             adminViewCustomerPayment()
172         else:
173             print("option not supported, try again")
174             adminOption()
175     elif choice == 4:
176         print("Please select a search option: \n a. Search Customer \n b. Search Customer's payment")
177         option = str(input("Enter your option: "))
178         if option == "a":
179             adminSearchCustomerInfo()
180         elif option == "b":
181             adminSearchCustomerPayment()
182         else:
183             print("option not supported, try again")
184             adminOption()
185     elif choice == 5:
186         exitPage()
187     else:
188         print("option not supported, try again")
189         adminOption()
190
```

After the admin have logged in, they are redirected to the `adminOption()` function which is defined at `line_149`. This function will print a menu consisting of 5 available options: `add event`, `modify event`, `display records`, `search records`, and `exit`. If user selects the first or second option, they will be redirected to the `addEvent()` and `modifyEvent()` functions accordingly. If, however, the user chooses third options another menu will be printed with 4 available options: `view category`, `view events`, `view customers`, view `customer payments`, which will branch out to `adminViewCategory()`, `adminViewEvent()`, `adminViewCustomerList()`, `adminViewCustomerPayment()` functions accordingly. Similarly, if users pick the fourth option, a search menu will be displayed with 2 two features: `search customer` and `search customer payment`, which calls the `adminSearchCustomerInfo()` and `adminSearchCustomerPayment()` functions accordingly. If at any point during the execution the user inserts an invalid option, the `adminOption()` function will be re-called for the user to retry. Finally, users can choose to end the AEGIS application using the fifth option, which will execute the `exitPage()` function.

> def addEvent()

```
192 def addEvent():
193     eventHandler = open("eventList.txt", "r")
194     print(">>> Welcome to AEMS event page, please select your category of events to add <<<")
195     while True: # while loop is used to display the available categories for admin if they decide to add another event later on
196         print("Event Category: \n 1. Sports/Martial Arts \n 2. Musical Performance \n 3. Dance/Performing Arts \n 4. Expo/Exhibition \n 5. Competition/Tournament \n 6. Others")
197         choice = int(input("Please enter your choice: "))
198         if choice == 1:
199             eventCat = "Sports & Martial Arts"
200             break
201         elif choice == 2:
202             eventCat = "Musical Performance"
203             break
204         elif choice == 3:
205             eventCat = "Dance & Performing Arts"
206             break
207         elif choice == 4:
208             eventCat = "Expo & Exhibition"
209             break
210         elif choice == 5:
211             eventCat = "Competition & Tournament"
212             break
213         elif choice == 6:
214             newCat = str(input("Enter the name of your category: "))
215             eventCat = newCat
216             break
217         else:
218             print("please select a valid option")
219             continue
220
```

The `addEvent()` function is defined at `line_192`. As this is a lengthy code, it will be divided into four parts for explanation. The first part (`line_193 - line_220`) consist of a file handler called `eventHandler` that `read` from the `eventList.txt` database. Then, a menu-driven display will be presented to the admin with all available event categories set by AEMS. The admin will then select the desired category from the menu by inputting an integer value in the terminal. This value will later be ran in an `if-else` statement – if the value is between 1-5, then the corresponding event category will be saved under the variable named `eventCat`; if the value is 6, the admin will be prompted to insert their own custom category name (e.g., Meet & Greet) which will be saved under the variable named `newCat` and later copied to `eventCat`. Otherwise, if the admin entered any other values, the system would print "Please select a valid option" and use the `continue` clause to loop back the to the `while` loop located at `line_195` for retry.

```
221     # to obtain user inputs
222     eventName = str(input("Enter event name: "))
223     eventOrg = str(input("Enter event organizer: "))
224     eventDate = str(input("Enter event start date in this format > [dd/mm/yyyy]: "))
225     eventTime = str(input("Enter event start time in 24-hour format [eg: 16:30]: "))
226     eventDuration = str(input("Enter event duration [eg: 2 days]: "))
227     eventLocation = str(input("Enter the venue of the event: "))
228
229     eventName = eventName.title() # for neatness/presentability purposes
230     eventLocation = eventLocation.title() # for neatness/presentability purposes
231
232     while True: # looping while to ask user whether event is free or not
233         eventFee = str(input("Is your event free-of-charge? [Y/N]: "))
234         if eventFee == "Y":
235             eventPrice = "0.00"
236             break
237         elif eventFee == "N":
238             eventPrice = str(input("Enter event price (per person) in RM [eg: 2.00]: ")) #string because we will concatenate it later (float will cause error)
239             break
240         else:
241             print("Error: Please insert either [Y] or [N]")
242             continue
243
```

After choosing the event category, admin will be required to fill-in the event name, organizers, date, time, duration, and location inside the terminal which will all be saved as a `string` format

for `concatenation` later on. The event name and location will be saved in a title format using the `title()` function. A second `while` loop is utilized at `line_232` to ask admins whether the event is free-of-charged using an `if-else` statement – if yes [Y], '0.00' will be saved into the variable named `eventPrice` as a `string`; if no [N], the admin will be required to fill-in the participation fee of the event, which will be saved under `eventPrice` as a `string` as well; otherwise, the system will prompt user to insert either [Y] or [N] in the terminal, and use the `continue` clause to re-loop the fee process.

```
244    # defining empty list
245    eCat = []
246    eName = []
247    eOrg = []
248    eDate = []
249    eTime = []
250    eDura = []
251    eLoca = []
252    eFee = []
253
254    # appending individual items into a list
255    for i in eventHandler:
256        a,b,c,d,e,f,g,h = i.split(", ")
257        h = h.strip()
258        eCat.append(a)
259        eName.append(b)
260        eOrg.append(c)
261        eDate.append(d)
262        eTime.append(e)
263        eDura.append(f)
264        eLoca.append(g)
265        eFee.append(h)
266
267    # appending content of each list into the text document
268    db = open("eventList.txt", "a")
269    db.write(eventCat+", "+eventName+", "+eventOrg+", "+eventDate+", "+eventTime+", "+eventDuration+", "+eventLocation+", "+eventPrice+"\n")
270    print("Registration Successful!")
271    db.close()
272    eventHandler.close()
```

Next, 8 empty list will be created with the name `eCat`, `eName`, `eOrg`, `eDate`, `eTime`, `eDura`, `eLoca`, `eFee` as the list name at (`line_245 - line_252`). After that, a `for` loop is used to copy all items in `eventHandler`, separate them using the `split()` function with '`, `' as the parameter, and save them into variables lettered '`a`' to '`h`' with h being stripped of all blank spaces using the `strip()` function. Values from these letters are later appended to the associating list at (`line_258 - line 265`). A new file handler `db` is used to open the `eventList.txt` database in `append` mode using the `open()` function, and the `.write()` is used to write each of the 8 values into the `eventList.txt` file using string concatenation with '`\n`' clause at the end to instigate a new line in the database. Finally, the 'Registration Successful!' message is printed, and both 'db' and 'eventHandler' are closed to update the new values into the database using the `close()` function.

```
274     while True: # option to return to homepage or add another event
275         print("Do you want to [1] return to admin homepage, [2] add another event?")
276         option = int(input("Your choice: "))
277         if option == 1:
278             print("returned")
279             adminHome()
280         elif option == 2:
281             addEvent()
282         else:
283             print("please select another option")
284             continue
285         break
286
```

Finally, a third `while` loop is used in combination with an `if-else` statement to initiate an error-detection feature which allow admins to [1], return to homepage by calling the `adminHome()` function or [2], add another event using `addEvent()` function. Any other inputs will result in a warning message followed by the `continue` clause, which will loop the options for admins to choose again. Once that's all done, the `break` clause is used at the end to exit the `while` loop as the entire `addEvent()` function has served its purpose.

> def modifyEvent()

```
288 def modifyEvent():
289     eventdb = open("eventList.txt", "r")
290     counter = 1
291     print("Which event would you like to amend?")
292     for i in eventdb: # display all available events
293         a,b,c,d,e,f,g,h = i.split(", ")
294         print(counter,"|", b)
295         counter += 1
296     eventdb.close()
297 
298     selection = int(input("Your choice: ")) # obtain user choice
299     selection -= 1 # important: we minus 1 to get the index value (because index starts at 0)
300 
301     modifydb = open("eventList.txt", "r")
302     modList = modifydb.readlines() # readlines() let us transfer the data from files into a temporary list
303     editList = [] # empty string to be used later
304     newList = modList # newlist copies modlist (for later use)
305     tempIndex = modList[selection] # temp index finds the index of the selected event
306     modifydb.close()
307 
308     print("\n>>> Current Event Details:") # display the selected events for users to refer to when modifying content(s)
309     print(tempIndex)
310 
```

Similarly, the `modifyEvent()` function as declared in `line_288` consist of a lengthy code which will be divided into three sections for a more detailed explanation. First, `eventdb` is used to `open` the `eventList` text file in `read` mode, then a counter is declared with an `integer` value of '`1`'. The contents in eventdb will then be read using a `for` loop, which each of the 8 variables separated using the `split()` function. Subsequently, the `counter` and variable '`b`' (event name) will be printed in a cyclic manner with the `counter` incrementing by `1` every cycle until all events are printed, then the `eventdb` handler is closed using the `close()` function to end the process. After the admin have reviewed the list of events, they can choose the specific events they want to modify by selecting the `integer` value associated with the event, as printed using the aforementioned `counter`. This selection value will then be deducted by `1` so that we can find the position of that event inside the array using index later on. A new handler called `modifydb` is created which opens `eventList.txt` in `read` mode. The contents of `modifydb` are copied into a list called `modList` using the `readlines()` function. The `editList` empty list is declared here to be used later on. Then, the contents of modList are duplicated into `newList` which will be used to re-write the database later on as well. Finally, `tempIndex` is declared and used to store the chosen event details which is extracted at array location `selection` in `modList`, using the `[]` square bracket index feature with values in `selection` as its parameter. The handler is then closed and the full details of the selected event (price, organizers, etc) are printed out for admin references during the 'amendment phase', which will be covered in the next section.

```
while True: # feature to display available categories presets
    print("Select New Event Category: \n 1. Sports/Martial Arts \n 2. Musical Performance \n 3. Dance/Performing Arts \n 4. Expo/Exhibition \n 5. Competition/Tournament \n 6. Others")
    choice = int(input("Please enter your choice: "))
    if choice == 1:
        mod1 = "Sports & Martial Arts"
        break
    elif choice == 2:
        mod1 = "Musical Performance"
        break
    elif choice == 3:
        mod1 = "Dance & Performing Arts"
        break
    elif choice == 4:
        mod1 = "Expo & Exhibition"
        break
    elif choice == 5:
        mod1 = "Competition & Tournament"
        break
    elif choice == 6:
        modCat = input("Enter the name of your category: ")
        mod1 = modCat
        break
    else:
        print("option not supported, try again")
        continue

# obtain new details for the modified events
mod2 = input("New event name: ")
mod3 = input("New event organizer: ")
mod4 = input("New event start date [eg: dd/mm/yyyy]: ")
mod5 = input("New event start time [eg: 16:30]: ")
mod6 = input("New event duration [eg: 2 days]: ")
mod7 = input("New venue of the event: ")
mod8 = input("New event price: ")
```

The second portion of this function is dedicated for amendments. Admin may now change the event category, event name, event org, etc by selecting the categories in the `if-else` statement and inputting the associative changes into the other fields under a `while` loop. The purpose of the `while` loop is to facilitate potential errors in the inputted values and respond using `break` or `continue` clause accordingly. These amendments will then be stored in a temporary `mod` variable and appended in the next section.

```
    # replace all changes in empty list
    editList.append(mod1)
    editList.append(mod2)
    editList.append(mod3)
    editList.append(mod4)
    editList.append(mod5)
    editList.append(mod6)
    editList.append(mod7)
    editList.append(mod8+"\n")
    finalEdit = ', '.join(editList) #join function to combine all strings into one string

    # delete the old event at the specific index and replace it with a new event at the same index (modification)
    del newList[selection]
    newList.insert(selection, finalEdit)

    # save changes into the .txt file
    db = open("eventList.txt", "w")
    for x in newList:
        db.write(x)
    db.close()
    print("Modification Successful!")
    adminOption()

```

The final portion of the code is used to update the new changes into the `eventList` text file. Once the admins have made all their changes, `editList` will be appended with their modifications using `append()`, and the `join()` function is used to combine them into one large string called `finalEdit`. The old event will be deleted using the `del` statement, and the new

event is inserted at that old event array location to replace it. This is done using the `insert()` function with the index given by `selection` and values given by `finalEdit`. Finally, the handler db opens `eventList` in `write` mode with a `for` loop to re-write each content of `newList` onto the text file using the `write()` function. The db handler is eventually closed, and the 'Modification Successful!' message is displayed to inform the admin that the process was successful. Upon completion of the entire `modifyEvent()` function, the `adminOption()` function will be called to return the admin back to the menu.

> def adminViewCategory()

```
370 def adminViewCategory():
371     eventdb = open("eventList.txt", "r")
372     # category counters are declared here
373     counterCat1 = 0
374     counterCat2 = 0
375     counterCat3 = 0
376     counterCat4 = 0
377     counterCat5 = 0
378     counterCat6 = 0
379     for i in eventdb: # counter that display the total event in each cateogory
380         a,b,c,d,e,f,g,h = i.split(", ")
381         if "Sports & Martial Arts" in a:
382             counterCat1 += 1
383         elif "Musical Performance" in a:
384             counterCat2 += 1
385         elif "Dance & Performing Arts" in a:
386             counterCat3 += 1
387         elif "Expo & Exhibition" in a:
388             counterCat4 += 1
389         elif "Competition & Tournament" in a:
390             counterCat5 += 1
391         else:
392             counterCat6 += 1
393 
394     print("Here are all the available Categories:")
395     print("\n 1. Sports & Martial Arts\n", "Number of event in this Cat =", counterCat1)
396     print("\n 2. Musical Performance\n", "Number of event in this Cat =", counterCat2)
397     print("\n 3. Dance & Performing Arts\n", "Number of event in this Cat =", counterCat3)
398     print("\n 4. Expo & Exhibition\n", "Number of event in this Cat =", counterCat4)
399     print("\n 5. Competition & Tournament\n", "Number of event in this Cat =", counterCat5)
400     print("\n 6. Others\n", "Number of event in this Cat =", counterCat6)
401     eventdb.close()
402     adminOption()
403 
404 
```

AEGIS also contains the `adminViewCategory()` function, as defined at `line_370`. This function starts with a handler called `eventdb` that opens `eventList.txt` in `read` mode, with 6 counters named `counterCat1` to `counterCat6` all set to `0`. A `for` loop is used to scan the content in `eventdb` and separate each item using the `split()` function with '`, `' as its delimiter. An `if-else` statement is then used in combination with the `in` statement, which will look search for the respective categories in `eventdb` and increase its category counter by `1` whenever there's a match. The results are then printed, with each category separated by a `\n` clause to generate a new line for tidiness of display, along with the number of events found in that specific category as indicate using their respective category counter. Finally, the handler is closed and admins are returned to the admin menu by calling the `adminOption()` function.

> def adminViewEvent()

```
405 def adminViewEvent():
406     eventdb = open("eventList.txt", "r")
407     counter = 1
408     print("here are all the event & their details:")
409     for i in eventdb: # a clean ui that presents all available event in the eventlist.txt file
410         a,b,c,d,e,f,g,h = i.split(", ")
411         print(counter,"|", b, "\nCategory:", a, "\nOrganizer:", c, "\nDate:", d, "\nTime:", e, "\nDuration:", f, "\nVenue:", g, "\nPrice (RM):", h, "\n")
412         counter += 1
413     eventdb.close()
414     adminOption()
415
```

Next up, the `adminViewEvent()` function is defined at `line_405` which enables admins to view the details of each event. It starts by declaring `eventdb` as the file handler and opens `eventList.txt` in `read` mode. A `counter` is then set to `1`, and a short message – 'here are all the events & their details:' is printed using the `print()` function. After that, a for loop is used to scan and split the content of each line into 8 objects. Finally, another `print()` function is used to display the `counter`, followed by the details of the event in order of name, category, organizer, date, time, duration, venue, and price. `\n` clause is also constantly used throughout the previous statement for the purpose of improved clarity. The `counter` will increment by `1` for every item in `eventdb` and stop when there are none left. At last, the `eventdb` handler is closed using the `close()` function, and admin is returned back into the menu by calling `adminOption()`.

> def adminViewCustomerList()

```
417  def adminViewCustomerList():
418      memberdb = open("memberList.txt", "r")
419      counter = 1
420      print("here are all the members:")
421      for i in memberdb:
422          a,b = i.split(", ")
423          print(counter,"|", a)
424          counter += 1
425      memberdb.close()
426      adminOption()
427
```

Similarly, admins can choose to display the list of customers by calling the function: `adminViewCustomerList()` as defined at `line_417`. This function uses the `open()` function to open `memberList.txt` in `read` mode under the `memberdb` handler and set a `counter` to `1`. After that, it uses the `print()` function to print 'here are all the members:' as the first output of the function. Then, a `for` loop is used to scan each item in `memberdb` and split the content of each line into variable '`a`' and '`b`' using the `split()` function. The code at (`line_421 - line_424`) basically means `for` each item in `memberdb`, `print` the `counter` value, followed by a '`|`' divider, and the value of '`a`' (which is associated with member name). The counter is also increment by 1 before looping to the next item in queue. Finally, once all items have gone through the `for` loop, `memberdb` will be closed using `close()` function, and admins will be returned to the main menu using the `adminOption()` function.

> def adminViewCustomerPayment()

```
429 def adminViewCustomerPayment():
430     paymentdb = open("cartList.txt", "r")
431     counter = 1
432     print("here are all the members:")
433     for i in paymentdb:
434         a,b,c = i.split(", ")
435         print(counter,"|", "\nMember Name:", a, "\nEvent Registered:", b, "\nPayment details:", c)
436         counter += 1
437     paymentdb.close()
438     adminOption()
439
```

The `adminViewCustomerPayment()`, as defined in `line_429`, is the final display pre-sets that an admin can choose from in `adminOption()`. This function starts by opening `cartList.txt` in `read` mode under the `paymentdb` handler. Then a `counter` is set to `1`, with the `string` titled: 'here are all the members:' printed as the first output. Then, a `for` loop is used to scan and `split` objects of each line into 3 parts using '`, `' as the delimiter. Next, the `print()` function is used to display `counter`, member name, event registered, and payment details using variables `a`, `b`, `c` with `\n` in between to improve the neatness of the output. Note that the `counter` is incremented by `1` using the arithmetic operation '`+=`' whenever it loops. Before ending the function, `paymentdb` is closed, and admins are returned to the main menu by calling `adminOption()`.

> def adminSearchCustomerInfo()

```
441 def adminSearchCustomerInfo():
442     search = input("Enter customer username to search: ")
443     searchdb = open("memberList.txt", "r")
444     searchList = searchdb.readlines()
445     flag = False
446     searchdb.close()
447     for i in range(0, len(searchList)): # looping algorithm to find the sequence where memberName is present
448         searchTemp = searchList[i].split(", ")
449         if search in searchTemp:
450             flag = True
451         else:
452             pass
453 
454     if flag:
455         print(search, "is a member of Asian Event Management Service")
456     else:
457         print(search, "is NOT a member of Asian Event Management Service")
458     adminOption()
459 
```

If admins chose option 4a in `adminOption()`, they will be redirected here, to `adminSearchCustomerInfo()` as defined at `line_441`. This function allows admin to search for the presence of a user in the `memberList.txt` database, indicating that they are/aren't a registered member of AEMS. The function starts by accepting a `search` value given by the admin, then `searchdb` is used to `open` `memberList.txt` in `read` mode, with the contents copied onto `searchList` with the `readlines()` function. Then, a `flag` value is set to `False` by default, and the handler `searchdb` is closed using the `close()` function. After that, a `for` loop is used with the `range()` function from range `0` to the length of `searchList` determined using the `len()` function. `searchTemp` is then declared to store each item in `searchList` at index `i` whilst splitting them based on the delimiter of '`, `' using the `split()` function. An `if-else` statement is then ran to check whether the username searched by the admin is present in the `searchTemp` list. If the username is present, the `flag` value is set to `True`; if username is not present, the flag value will remain `False` and the `pass` clause will be used as a null statement to safely end the `if-else` process. Another `if-else` statement is deployed at `line_454` which will print whether or not the searched username is a part of the AEMS database. At the end of this function, admins will be brought back to the main menu using `adminOption()`.

> def adminSearchCustomerPayment()

```
461 def adminSearchCustomerPayment():
462     search = input("Enter customer username to search payment details: ")
463     search = search.capitalize() # all usernames are saved with capitalize in events by default
464     checkIndexList = []
465     a = '1' # these are used as positive flags
466     b = '0' # these are used as negative flags
467     print("\n=== Here are the payment details for", search, "===")
468     checkdb = open("cartList.txt", "r")
469     checkList = checkdb.readlines()
470     checkdb.close()
471     for i in range(0, len(checkList)): # looping algorithm to find the sequence where memberName is present
472         tempCheck = checkList[i].split(", ")
473         if search in tempCheck:
474             checkIndexList.append(a)
475         else:
476             checkIndexList.append(b)
477     for i in range(0, len(checkIndexList)):
478         if '1' in checkIndexList:
479             x = checkIndexList.index('1')
480             print(checkList[x].replace('\n',''))
481             checkIndexList[x] = '0'
482         else:
483             pass
484     adminOption()
485
```

The `adminSearchCustomerPayment()` function, as defined at `line_461` is called when admins select 4a as their choice in `adminOption()`. This function starts by prompting the admin to insert a username into a search bar, which will be saved under a `string` format within the variable named '`search`'. As the first letter of the username saved in the cartList.txt database is capitalized, the program will automatically capitalize the '`search`' input using the `capitalize()` function. Next, an empty list is made with the name '`checkIndexList`' and variables '`a`' and '`b`' are declared and used to store string content '`1`' and '`2`' as positive and negative flags respectively. The `print()` function is then deployed to display a message which reads: "Here are all the payment details for [searched username]" with `\n` as prefix to initiate a new line and separate the display from the input box for enhanced readability. After that, the handler '`checkdb`' is used to `open` `cartList.txt` in `read` mode, with its content copied into a list named '`checkList`' using the function `readlines()`. Once `checkList` is filled with content, the `checkdb` handler can now be safely closed using the `close()` function. A `for` loop is initiated to scan the contents from range 0 until the full-length of checkList using the `range()` and `len()` function respectively. A '`tempCheck`' variable is then declared and used to store each individual items in `checkList` by splitting the list into smaller, individual strings using `split()` function with '`, `' as its delimiter. An `if-else` statement is then deployed to search whether the username searched is present in `tempCheck`, here's how it works: whenever the username is present, the flag '`1`' will be `appended` to `checkIndexList` using the variable '`a`'; whenever the username is

NOT present, the flag ‘0’ will be `appended` to `checkIndexList` using the variable ‘b’. The final result is a sequence in the form of [‘0’, ‘0’, ‘1’, ‘1’, ‘0’…] which means [‘absent in index 0’, ‘absent in index 1’, ‘present in index 2’, ‘present in index 3’, ‘absent in index 4’, and so on… The goal of this algorithm is to produce a sequence that can be used to identify the specific location of the username inside the `eventList.txt` database. Once we have the sequence, another `for` loop is used to decipher the results. This time, AEGIS uses the `index()` function with the value ‘`1`’ as its parameter. Essentially, this will return the location of ‘`1`’s inside the checkIndexList array (e.g., [1], [4], [5]), which are saved into variable ‘`x`’. After the `index(‘1’)` is identified, it is replaced to ‘`0`’ using the `replace()` function, so that the location of the next ‘`1`’ can be identified in the list. The final result ‘`x`’ is then printed which will contain the searched username as inserted by the admin from the start, with its associated events and payment details. When all the payment details under that username are printed, the program safely ends with a `pass` clause acting as a null statement to break out of the `if-else` statement, and admins are returned to the main menu by calling `adminOption()`.

> def guestViewEvent()

```
487 def guestViewEvent():
488     eventdb = open("eventList.txt", "r")
489     counter = 1
490     print("\nhere are all the event & their details: \n")
491     for i in eventdb:
492         a,b,c,d,e,f,g,h = i.split(", ")
493         print(counter,"|", b, "\nCategory:", a, "\nOrganizer:", c, "\nDate:", d, "\nTime:", e, "\nDuration:", f, "\nVenue:", g, "\nPrice (RM):", h, "\n")
494         counter += 1
495     eventdb.close()
496     back = input("press <enter> to return")
497     if back == "":
498         guestOption()
499     else:
500         guestOption()
501
502
```

Aside from admin, AEGIS also have many features for guest who do not wish to register as member. The first is the `guestViewEvent()` function, which is defined at `line_487`. This function begins by using the `open()` function to open `eventList.txt` inside the handler called `eventdb`, with a `counter` set to '`1`'. A `print()` function is then used to print out: 'here are all the event & their details:' with a prefix and subfix of `\n` for improved tidiness. Then, a `for` loop is deployed to look through each item in eventdb and organized them into 8 parts using the `split()` function with '`, `' as its parameter. Each event is then display using a `counter` followed by a sequence of name, category, organizer, date, time, duration, venue, and price. The `counter` is incremented by '`1`' for every event listed. At the end, the handler `eventdb` will be closed using the `close()` function, and guests are given a choice to return to the main menu by pressing `<enter>` (or any values) into the terminal, which will call the `guestOption()` function.

> def guestViewCategory()

```
503  def guestViewCategory():
504      print("\nhere are the available event categories: \n")
505      print("1. Sports & Martial Arts")
506      print("2. Musical Performance")
507      print("3. Dance & Performing Arts")
508      print("4. Expo & Exhibition")
509      print("5. Competition & Tournament")
510      print("6. Others\n")
511      back = input("press <enter> to return")
512      if back == "":
513          guestOption()
514      else:
515          guestOption()
516
```

As the amount of event categories in the AEMS system are fixed, the `guestViewCategory()` function, as defined in `line_503`, simply uses the `print()` function to print out each of the 6 available categories for guest to behold. Guests are then asked to press `<enter>` or insert any value into the terminal, in which `guestOption()` will be summoned to bring them back to the main menu.

> def memberSignup()

```
518 def memberSignup():
519     memberHandler = open("memberList.txt", "r")
520     print(">>> Welcome to AEMS member registration page, please follow the instruction below <<<")
521     username = input("Enter a username: ")
522     password1 = input("Enter a password: ")
523     password2 = input("Confirm your password: ")
524 
525     # database
526     memberUN = []
527     memberPW = []
528     for i in memberHandler:
529         a,b = i.split(", ")
530         b = b.strip()
531         memberUN.append(a)
532         memberPW.append(b)
533 
534     if password1 != password2:
535         print("Password does not match, please try again")
536         memberSignup()
537     elif len(password1) <= 8:
538         print("Password must be longer than 8 characters, please try again")
539         memberSignup()
540     elif username in memberUN:
541         print("Username already taken, please try again")
542         memberSignup()
543     else:
544         db = open("memberList.txt", "a")
545         db.write(username+", "+password1+"\n")
546         print("Registration Successful!")
547         db.close()
548     memberHandler.close()
549     guestOption()
```

The `memberSignup()` function, as defined in `line_518`, contains a very similar format that is found inside the `adminSignup()` function at `line_88`. First, `memberHandler` is used to `open` `memberList.txt` in `read` mode, then a message is printed, asking users to enter their respective username and password into the given field. A series of code from (`line_526 - line_532`) basically look through the username and passwords inside the existing database, `splits` them, `strips` the password, and `append` them into two temporary lists called `memberUN` and `memberPW`. These two lists are then used as reference to find whether the username already exist within the system using an `if-else` statement paired with the `in` clause. The passwords are also compared to see if it matches, and if it exceeds 8-character length using the `len()` function. If any error were to occur, a message will be printed, and `memberSignup()` will be called again, guiding users back to the sign-up page to retry. Once all is successful, the `db` handler will `open` `memberList.txt` in `append` mode, and write the new username and password using the `write()` function. Finally, both `db` and `memberHandler` are closed, and guests are returned to the main menu, with a ‘Registration Successful!’ message printed to inform users that their registration details have been added to the AEMS database.

> def memberLogin()

The `memberLogin()` function, as defined in `line_552`, is by far the most complex code with the greatest amount of code density [140 lines of codes], so it will be split into 6 parts for explanation.

[Part 1] – `line_552-568`

```
552 def memberLogin():
553     dt_today = str(datetime.datetime.today())
554     tempList = []
555     memberHandler = open("memberList.txt", "r")
556     while True:
557         print(">>> Member Login Page <<<")
558         memberUN = input("Enter your username: ")
559         memberPW = input("Enter your password: ")
560         for i in memberHandler: # search database for users
561             tempList.append(i)
562         if memberUN + ", " + memberPW + "\n" in tempList:
563             print("Login successful!")
564             print("Welcome back,", memberUN.capitalize())
565             memberName = memberUN.capitalize() # we saved their username as memberName
566             cartItems = []
567             cartTotal = 0
568
```

`dt_today` is a variable declared to store the `datetime` value imported from the `datetime` library, summoned using the `today()` function. A `tempList` list is then declared which is an empty array that will be used later. `memberHandler` is also used as a file handler to `open` `memberList.txt` in `read` mode so it can be used later on as well. The first `while` loop [1/4] is instigated at `line_556` to facilitate any errors in the login page by allowing user to retry later on using `continue`. Users are then asked to login into the system by inserting their username into `memberUN` and password into `memberPW`. These credentials are then concatenated and compared with the memberList.txt data to see if the combination exists in the database. The comparison is done using a `for` loop to `append` each item of `memberHandler` into `tempList` as reference. An `if-else` statement will examine both the combination inserted and the reference given to decide whether the credentials are valid. If it's valid, it will print "Login Successful!" with a "Welcome back, [username]" message that is capitalized using the `capitalize()` function as saved as `memberName`. Finally, the `cartItems` empty list is declared and `cartTotal` is set to `0` for later use.

[Part 2] – `line_569-601`

```
569             while True:
570                 # option
571                 print("\nwhat would you like?")
572                 print("1. Add events to Shopping Cart")
573                 print("2. View purchased events")
574                 print("3. Exit")
575                 answer = int(input("Your Choice: "))
576 
577                 if answer == 1: #add + payment + write
578                     while True:
579                         counter = 1
580                         print("Which event would you like to add?")
581                         eventdb = open("eventList.txt", "r")
582                         for i in eventdb:
583                             a,b,c,d,e,f,g,h = i.split(", ")
584                             print(counter,"|", b)
585                             counter += 1
586                         eventdb.close()
587 
588                         # obtain user choice
589                         try:
590                             selection = int(input("Your choice: "))
591                             if selection in range(0, counter):
592                                 selection -= 1
593                                 pass
594                             else:
595                                 print("\n>>> selected value out of bound, try again\n")
596                                 continue
597 
598                         except ValueError:
599                             print("\n>>> please insert an integer value\n")
600                             continue
601 
```

The second `while` loop [2/4] in deployed within the first while loop to act as an error detection system so users can retry when they have inserted an invalid value. This section contains an option menu which will `print` 3 available options for the member to select. If the member chooses the first option, the member will be brought to the add-cart-section of the code, as identified using a third `while` loop [3/4] at `line_578` as a multi-cart adding system, with a `counter` set to `1`. The file handler `eventdb` will then `open` `eventList.txt` in `read` mode, `split` the list into 8 items, and `print` each event categories out one-by-one with an incrementing counter value of 1. The `eventdb` is then closed with the `close()` function, and the user is asked to select an event to be added into their shopping cart, which will be an integer value stored in `selection`. The `try-except` statement is used to prevent crashing the system if user enters a string value instead of an integer value by allowing them to retry using the `continue` clause. The `if-else` statement at `line_591` prevents crashes if users were to enter values which are out of the selection criteria by allowing to retry using the `continue` clause. Otherwise, if the proper

value is inserted, the selection value is deducted by 1 (to find array later on), and users are brought to part 3 of this program (see below).

[Part 3] – `line_602-622`

```
602                 # open the eventlist to extract that specific event
603                 itemdb = open("eventList.txt", "r")
604                 itemList = itemdb.readlines()
605                 itemIndex = itemList[selection].split(", ")
606                 addItem = itemIndex[1]
607                 itemPrice = itemIndex[7].replace('\n','') # prevents double spacing
608                 itemPrice = float(itemPrice)
609                 itemdb.close()
610
611                 cartItems.append(addItem)
612                 cartTotal += itemPrice
613                 choose = input("do you want to add more items? <Y/N>")
614                 if choose == "Y":
615                     continue
616                 elif choose == "N":
617                     print("proceeding to payment...")
618                     break
619                 else:
620                     print("option not supported, proceeding to payment")
621                     break # add cart
622
```

Next, `itemdb` is used to open `eventList.txt` in `read` mode, and the contents of `itemdb` are copied into `itemList` using `readlines()`. Then, the chosen event will be selected using the `selection` value in `square brackets []` in order to locate it inside the `itemList` array through indexing. After extracting the event, it is saved into `itemIndex` with the event details `split` using '`, `' as delimiter. The event name (located at `itemIndex[1]`) and the event price (located at `itemIndex[7]`) are extracted and saved under `addItem` and `itemPrice` respectively. The itemPrice will have '\n' replaced with nothing using the `replace()` function, so that it can be converted into a float using the `float()` function. Finally, the `itemdb` is closed, as it is not needed anymore. Now, the `cartItems` list declared at the start can be used to `append` all the events chosen by the user, while the `cartTotal` is used to sum up the total price of each `itemPrice` using the arithmetic operation '`+=`'. Members will then be asked if they want to <Y> add more events, which will bring them back to the third while loop at [3/4] at `line_578` using `continue`, or <N> proceeds to payment, which will bring them to Part 4, payment system, using `break`.

[Part 4] – `line_623-662`

```
623                 while True:
624                     print("total amount dued: ", cartTotal)
625                     try:
626                         payment = float(input("please enter your payment amount: "))
627                         if payment == cartTotal:
628                             print("payment successful!")
629                             summaryPrice = str(cartTotal)
630                             balance = 0.00
631                             break
632                         elif payment < cartTotal:
633                             print("insufficient amount, please try again")
634                             continue
635                         elif payment > cartTotal:
636                             balance = payment - cartTotal
637                             print("payment successful! RM", balance, "have been returned to your account")
638                             summaryPrice = str(cartTotal)
639                             break
640                         else:
641                             print("please insert a valid number")
642                             continue # payment
643                     except ValueError:
644                         print("\n>>> please insert a valid amount!\n")
645                         continue
646 
647                 print("\nOrder Summary:")
648                 print("You bought:", cartItems)
649                 print("Total cost:", cartTotal)
650                 print("You paid:", payment)
651                 print("Balance returned:", balance)
652                 print("order completed! thank you for your support!")
653                 print('')
654                 eventdb.close()
655 
656                 cartdb = open("cartList.txt", "a")
657                 for i in cartItems:
658                     cartdb.write(memberName+", "+i+", "+"paid on "+dt_today+"\n")
659                 cartdb.close()
660                 continue
661 
662 
```

Part 4 features the fourth and final `while` loop that would also act as a balance-checking system, with a `try-except` statement acting as a `ValueError` detection system. At this point, the member has already confirmed the cart, and awaiting payment. Thus, the 'total amount dued: ' will be printed with `cartTotal` as its total price. An input box will then be displayed for users to input the amount payable, which will be saved under a float value. If the user attempts to enter a string value, the `except` statement will be triggered, which ask user to enter a valid option, and bring user back to the fourth `while` loop using `continue`. If the user attempts to enter a `payment` amount that is below the `cartTotal` amount, or no amount at all, they will be asked to retry using the `continue` clause. If the user paid the correct `payment` amount, or any amount above the `cartTotal`, the payment process will be successful, and the `balance` will be calculated and returned to the user, and the `break` clause will be used to exit the `while` loop. Once the payment is done, an order summary is displayed with `cartItems`, `cartTotal`, `payment`, `balance`, and a

message thanking the user for their patronage. The last thing in payment is to write the name of the member, their purchased events and transaction date into `cartList.txt` as opened by `cartdb` in `append` mode. The cartdb is then saved with the close() function, and members are returned to the second `while` loop [2/4] at `line_569`.

[Part 5] – `line_663-686`

```
663             elif answer == 2:
664                 checkIndexList = []
665                 a = '1' # these are used as positive flags
666                 b = '0' # these are used as negative flags
667                 print("\n=== Here are all your registered events ===")
668                 checkdb = open("cartList.txt", "r")
669                 checkList = checkdb.readlines()
670                 checkdb.close()
671                 for i in range(0, len(checkList)): # looping algorithm to find the sequence where memberName is present
672                     tempCheck = checkList[i].split(", ")
673                     if memberName in tempCheck:
674                         checkIndexList.append(a)
675                     else:
676                         checkIndexList.append(b) # output is a list like this [1,0,0,1,0,1] where 1 means member is present in record
677 
678                 for i in range(0, len(checkIndexList)):
679                     if '1' in checkIndexList:
680                         x = checkIndexList.index('1') # with the algorithm, we can simply print the line where the corresponding memberName is present
681                         print(checkList[x].replace('\n','')) # this is to remove to 'new line' feature to prevent double spacing during output
682                         checkIndexList[x] = '0' # then we replace the '1' in the algorithm to '0' after printing the values
683                     else:
684                         pass
685                 continue
686 
```

Part 5 is codes that will be executed if users choose option number 2. This function contains the exact flag-searching algorithm as covered in detail in `adminSearchCustomerPayment()`. Essentially, '`a`' and '`b`' are used to store '`1`' and '`2`' as 'positive' and 'negative' flags. Then a checking-algorithm as covered in `adminSearchCustomerPayment()` is used to produce a sequence of ['0', '1', '0', '0'…] where '0' means the user is not present, and '1' means the user is present in that specific line. The `index` of that specific line containing the matching member name is then returned and printed out as the output. Finally, after reading the events that they have purchased, the members are returned back to the second `while` loop [2/4] where they can choose other options.

[Part 6] – `line_687-706`

```
687                 elif answer == 3:
688                     exitPage()
689 
690                 else:
691                     print("choice not supported, try again")
692                     continue
693 
694                 break
695 >           else:
705             break
706     memberHandler.close()
707 
708 
```

If the member chooses option [3] Exit, the `exitPage()` function will be called, ending the AEGIS program. However, if the user enters any other `integer` values other than 1, 2, or 3, they will be asked to retry using the `continue` clause to the second `while` loop [2/4]. Otherwise, the AEGIS program will break out of the second while loop [2/4] at `line_692` and break out of the first `while` loop [1/4] at `line_705`, and eventually closing `memberHandler` using the `close()` function.

> def exitPage()

```
709  def exitPage():
710      print("\nYou have exited the application")
711      print("Thank you for using our services!")
712
713
```

The `exitPage()` function as defined in `line_709` is a simple message thanking the user for spending time using AEGIS on the AEMS website. It also acts as a definitive way to end the entire AEGIS program without crashing it, thus marking the ending of the entire program.

> homepage()

```
714  homepage() #initiate this upon launching AEGIS
```

The `homepage()` function is called last at `line_714`, which will allow AEGIS to start running when it is booted in the terminal.

Additional Features of Source Code with Explanation

> Using while loop for error prevention

```
while True:
    print("total amount dued: ", cartTotal)
    try:
        payment = float(input("please enter your payment amount: "))
        if payment == cartTotal:
            print("payment successful!")
            summaryPrice = str(cartTotal)
            balance = 0.00
            break
        elif payment < cartTotal:
            print("insufficient amount, please try again")
            continue
        elif payment > cartTotal:
            balance = payment - cartTotal
            print("payment successful! RM", balance, "have been returned to your account")
            summaryPrice = str(cartTotal)
            break
        else:
            print("please insert a valid number")
            continue # payment
    except ValueError:
        print("\n>>> please insert a valid amount!\n")
        continue
```

There will be instances where a user will enter an undesirable input, such as choosing '7' from the range of 1-6. In these scenarios, the application may not know how to react, causing it to produce an error, and exits the application entirely. However, AEGIS solves this by practicing the use of `while` loop as an error-checking method across many different functions. Users can easily be brought back to retry their inputs using `continue` if they have mistakenly provided an invalid input. The `break` statement can also be used in conjunction to proceed to the next stage if a valid input is obtain afterwards. In short, AEGIS not only uses `while` loop to print statements, but also uses it to prevent crashes and improve user experience.

> Admin can numbers of events in each category

```
def adminViewCategory():
    eventdb = open("eventList.txt", "r")
    counter = 1
    counterCat1 = 0
    counterCat2 = 0
    counterCat3 = 0
    counterCat4 = 0
    counterCat5 = 0
    counterCat6 = 0
    for i in eventdb: # counter that display the total event in each cateogory
        a,b,c,d,e,f,g,h = i.split(", ")
        if "Sports & Martial Arts" in a:
            counterCat1 += 1
        elif "Musical Performance" in a:
            counterCat2 += 1
        elif "Dance & Performing Arts" in a:
            counterCat3 += 1
        elif "Expo & Exhibition" in a:
            counterCat4 += 1
        elif "Competition & Tournament" in a:
            counterCat5 += 1
        else:
            counterCat6 += 1

    print("Here are all the available Categories:")
    print("\n 1. Sports & Martial Arts\n", "Number of event in this Cat =", counterCat1)
    print("\n 2. Musical Performance\n", "Number of event in this Cat =", counterCat2)
    print("\n 3. Dance & Performing Arts\n", "Number of event in this Cat =", counterCat3)
    print("\n 4. Expo & Exhibition\n", "Number of event in this Cat =", counterCat4)
    print("\n 5. Competition & Tournament\n", "Number of event in this Cat =", counterCat5)
    print("\n 6. Others\n", "Number of event in this Cat =", counterCat6)
    eventdb.close()
    adminOption()
```

AEGIS allows admin to not only view the event category, but also the **numbers** of events in each category as well. This extra feature allows AEMS admins to have a better overall knowledge on which event categories are more popular and least popular, so they can adjust their business plans and make future market trend predictions accordingly.

> User can directly sign up if login details are incorrect

```
        else:
            print("\n >>> invalid credentials... \n")
            option = input("would you like to sign-up? \n [Y] for admin registration \n [N] to retry log
            if option == "N":
                continue #repeat while loop
            elif option == "Y":
                adminAuth()
            else:
                print("invalid option, returning to homepage...")
                homepage()
        break
    adminHandler.close()
```

When a user (admin/member) attempts to login but have foreign credentials that are not present in the database, they will be given an option to directly signup as an admin/member without needing to return to homepage and re-select the process again. This will help to save time and allow users to have a more satisfactory experience and easier time using AEGIS.

> try and except crash prevention system

```
try:
    selection = int(input("Your choice: "))
    if selection in range(0, counter):
        selection -= 1
        pass
    else:
        print("\n>>> selected value out of bound, try again\n")
        continue

except ValueError:
    print("\n>>> please insert an integer value\n")
    continue
```

In cases where an `integer` or `float` value is required from the `input`, if a string were to be mistakenly entered, it will produce a `ValueError` and crash the system. This issue is resolved using the `try-except statements` where AEGIS will attempt to check for the **type** of issues that had occurred, and if it's a `ValueError` (such as inserting string into an integer-only input), it will produce a message telling user to 'please insert an integer value', instead of crashing the program entirely based on a small accident by the user. This `try-except` crash prevention system feature is especially important during the **payment** phase because if the program crashed, the user would have to add events to cart all over again and purchase them all over again, causing unnecessary frustration and wasting time. Therefore, this feature is essential in making AEGIS a hassle-free event management solution for Asian Event Management Service (AEMS).

Screenshots of Sample Input / Output with Explanation

Homepage – first thing upon launching AEGIS

```
=== Welcome to Asian Event Management Service (AEMS) ===
Please select your position:
1. Admin
2. Customer
3. Exit
Please select an option:
```

This is what the users will see when they first boot up AEGIS for the first time.

Admin Option

```
=== Please select an option below: ===
1. Admin Login
2. Admin Registration
3. Go Back
Your choice:
```

When the user selects [1] from homepage, they will be brought to admin option.

Admin Login (invalid credentials)

```
>>> Admin Login Page <<<
Enter your username: richard
Enter your password: bonsaitree

 >>> invalid credentials...

would you like to sign-up?
 [Y] for admin registration
 [N] to retry login
 Your choice:
```

When admin inserts an invalid credentials (example: `richard`, `bonsaitree`), they are given the choice to [Y] sign up, or [N] retry login.

```
# Admin Login (login successful)
```

```
>>> Admin Login Page <<<
Enter your username: amardeep
Enter your password: ilovepython
Login successful!
Welcome back, Amardeep

 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice:
```

When admin inserts the correct credentials (`amardeep`, `ilovepython`), 'Login successful!' is printed, and a greeting is used to welcome the user.

```
# Admin Authentication
```

```
=== Please select an option below: ===
1. Admin Login
2. Admin Registration
3. Go Back
Your choice: 2
Please enter company referral code:
```

When user chooses to sign-up as admin, they will be required to go through an authentication process.

```
# Admin Authentication (incorrect referral code)
```

```
Please enter company referral code: peanut
Incorrect referral code.
do you want to (1) Try Again, or (2) Go Back ?
Your choice:
```

When an incorrect referral code is inserted, an error will be printed, and users can choose between (1) Try Again, or (2) Go Back.

Admin Authentication (correct referral code)

```
Please enter company referral code: AEMS0001
Verification successful!
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username:
```

When the correct referral code is inserted, the system will congratulate the user, and allow them to sign-up as admin.

Admin Registration (username exist)

```
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username: amardeep
Enter a password: ilovejava
Confirm your password: ilovejava
Username already taken, please try again
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username:
```

When a user attempts to register using a username that is already taken, an error will be printed and users will be prompted to retry.

Admin Registration (password does not match)

```
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username: richard
Enter a password: watermelon
Confirm your password: durian
Password does not match, please try again
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username:
```

When a user enters passwords that does not match, an error will be printed and users will be prompted to retry.

Admin Registration (password too short)

```
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username: richard
Enter a password: mango
Confirm your password: mango
Password must be longer than 8 characters, please try again
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username:
```

When a user enters passwords that are below 8 characters, an error will be printed and users will be prompted to retry.

```
# Admin Registration (successful)
```

```
>>> Welcome to AEMS admin registration page, please follow the instruction below <<<
Enter a username: richard
Enter a password: bonsaitree
Confirm your password: bonsaitree
Registration Successful!
=== Please select an option below: ===
1. Admin Login
2. Admin Registration
3. Go Back
Your choice:
```

adminList.txt

```
1  dalton, ilovealmond
2  amadea, peanutbutter
3  amardeep, ilovepython
4  akansha, ilovedonuts
5  ethan, octoplush
6  richard, bonsaitree
7
```

When a user successfully passed all of the constraint checks, their username and password will be added into the adminList.txt database.

```
# Admin Add Events (full process)
```

```
Login successful!
Welcome back, Richard

 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 1
>>> Welcome to AEMS event page, please select your category of events to add <<<
Event Category:
 1. Sports/Martial Arts
 2. Musical Performance
 3. Dance/Performing Arts
 4. Expo/Exhibition
 5. Competition/Tournament
 6. Others
Please enter your choice: 1
Enter event name: Karate Sparring Competition (KSC)
Enter event organizer: Mahsa Karate Club
Enter event start date in this format > [dd/mm/yyyy]: 20/10/2022
Enter event start time in 24-hour format [eg: 16:30]: 12:00
Enter event duration [eg: 2 days]: 8 hours
Enter the venue of the event: Mahsa University - Hall 2
Is your event free-of-charge? [Y/N]: N
Enter event price (per person) in RM [eg: 2.00]: 10
Event added Successfully!
Do you want to [1] return to admin homepage, [2] add another event?
Your choice:
```

When admin choose to add an event, they will need to select a category from the list, insert the event name, organizer, date, time, duration, venue and whether the event is free-of-charged. In this scenario, Richard (the admin) wants to add a Karate Sparring Competition (KSC) as a new event which will take place on 20th of October, and by following the step-by-step instruction, he is able to append his new event into the eventList.txt file, as shown below.

All the information regarding KSC are now saved on a `line_8`. Richard can now choose to [1] return to the admin homepage or [2] add another event, which will continue appending events onto this eventList.txt file.

Modify Event (full process)

testrun.py eventList.txt

```
1  Sports & Martial Arts, Taekwondo Poomsae Showcase, APU Taekwondo Club, 04/06/2022, 13:00, 2 hours, APU Atrium, 10.00
2  Dance & Performing Arts, Battlegrounds 2022, Asia Momentum Media, 21/12/2022, 10:00, 3 days, DWI Emas Internation School - Auditorum 2, 5.00
3  Musical Performance, Open Mic 2022, Sunway Music Association, 07/07/2022, 18:00, 4 hours, Sunway Pyramid Blue Atrium, 0.00
4  Expo & Exhibition, Education Fair KLCC, MOE Malaysia, 05/11/2022, 14:00, 4 days, Suria KLCC, 0.00
5  Competition & Tournament, VALORANT Masters, Todak Academy, 06/11/2022, 15:00, 3 days, Todak Discord Server, 0.00
6  Meet & Greet, Jay Chow Fan Meeting, TVB, 14/10/2022, 17:00, 5 hours, Bukit Jalil Stadium, 100.00
7  Expo & Exhibition, Pet Expo 2022, WWF, 12/10/2022, 14:00, 2 days, Midvalley Megamall Convention Centre, 5.00
8  Sports & Martial Arts, Karate Sparring Competition (Ksc), Mahsa Karate Club, 20/10/2022, 12:00, 8 hours, Mahsa University - Hall 2, 10
9
```

Let's say Richard would like to amend the "Battleground 2022" event located at `line_2`, because he realized that it was set on the wrong date, with the wrong price. Here's how he can modify the event using AEGIS.

```
Login successful!
Welcome back, Richard

 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 2
Which event would you like to amend?
1 | Taekwondo Poomsae Showcase
2 | Battlegrounds 2022
3 | Open Mic 2022
4 | Education Fair KLCC
5 | VALORANT Masters
6 | Jay Chow Fan Meeting
7 | Pet Expo 2022
8 | Karate Sparring Competition (Ksc)
Your choice: 2

>>> Current Event Details:
Dance & Performing Arts, Battlegrounds 2022, Asia Momentum Media, 21/12/2022, 10:00, 3 days, DWI Emas Internation School - Auditorum 2, 5.00

Select New Event Category:
 1. Sports/Martial Arts
 2. Musical Performance
 3. Dance/Performing Arts
 4. Expo/Exhibition
 5. Competition/Tournament
 6. Others
Please enter your choice: 3
New event name: Battleground 2022
New event organizer: Asia Momentum Media
New event start date [eg: dd/mm/yyyy]: 22/12/2022
New event start time [eg: 16:30]: 10:00
New event duration [eg: 2 days]: 3 days
New venue of the event: DWI Emas Intersation School - Audi 2
New event price: 15.00
Modification Successful!
```

By insert the new event details into the `modifyEvent()` function, Richard is able to modify the event at line_2 as shown below.

testrun.py eventList.txt

```
1  Sports & Martial Arts, Taekwondo Poomsae Showcase, APU Taekwondo Club, 04/06/2022, 13:00, 2 hours, APU Atrium, 10.00
2  Dance & Performing Arts, Battleground 2022, Asia Momentum Media, 22/12/2022, 10:00, 3 days, DWI Emas Intersation School - Audi 2, 15.00
3  Musical Performance, Open Mic 2022, Sunway Music Association, 07/07/2022, 18:00, 4 hours, Sunway Pyramid Blue Atrium, 0.00
4  Expo & Exhibition, Education Fair KLCC, MOE Malaysia, 05/11/2022, 14:00, 4 days, Suria KLCC, 0.00
5  Competition & Tournament, VALORANT Masters, Todak Academy, 06/11/2022, 15:00, 3 days, Todak Discord Server, 0.00
6  Meet & Greet, Jay Chow Fan Meeting, TVB, 14/10/2022, 17:00, 5 hours, Bukit Jalil Stadium, 100.00
7  Expo & Exhibition, Pet Expo 2022, WWF, 12/10/2022, 14:00, 2 days, Midvalley Megamall Convention Centre, 5.00
8  Sports & Martial Arts, Karate Sparring Competition (Ksc), Mahsa Karate Club, 20/10/2022, 12:00, 8 hours, Mahsa University - Hall 2, 10
9
```

Admin View Event Categories

```
What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 3
display options:
 a. Event Category
 b. All Events
 c. Registered Customers
 d. Customer Payment
Enter your option: a
Here are all the available Categories:

 1. Sports & Martial Arts
 Number of event in this Cat = 2

 2. Musical Performance
 Number of event in this Cat = 1

 3. Dance & Performing Arts
 Number of event in this Cat = 1

 4. Expo & Exhibition
 Number of event in this Cat = 2

 5. Competition & Tournament
 Number of event in this Cat = 1

 6. Others
 Number of event in this Cat = 1
```

If Richard chooses the option `[3] Display record details`, followed by `[a] Event Category`, the event categories will be printed as shown above, with the numbers of event in each category printed as well.

```
# Admin View Events
```

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 3
display options:
 a. Event Category
 b. All Events
 c. Registered Customers
 d. Customer Payment
Enter your option: b
here are all the event & their details:
1 | Taekwondo Poomsae Showcase
Category: Sports & Martial Arts
Organizer: APU Taekwondo Club
Date: 04/06/2022
Time: 13:00
Duration: 2 hours
Venue: APU Atrium
Price (RM): 10.00


2 | Battleground 2022
Category: Dance & Performing Arts
Organizer: Asia Momentum Media
Date: 22/12/2022
Time: 10:00
Duration: 3 days
Venue: DWI Emas Intersation School - Audi 2
Price (RM): 15.00


3 | Open Mic 2022
Category: Musical Performance
Organizer: Sunway Music Association
Date: 07/07/2022
Time: 18:00
Duration: 4 hours
Venue: Sunway Pyramid Blue Atrium
Price (RM): 0.00


4 | Education Fair KLCC
Category: Expo & Exhibition
Organizer: MOE Malaysia
Date: 05/11/2022
Time: 14:00
Duration: 4 days
Venue: Suria KLCC
Price (RM): 0.00


5 | VALORANT Masters
Category: Competition & Tournament
Organizer: Todak Academy
```

If Richard chooses option `[3] Display record details`, followed by `[b] All Events`, each event on the `eventList.txt` file will be printed out in a neat display.

*Note that there are more events than this, but the screenshot is too small to include all of them.

```
# Admin View Customer
```

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 3
display options:
 a. Event Category
 b. All Events
 c. Registered Customers
 d. Customer Payment
Enter your option: c
here are all the members:
1 | zorus
2 | florence
3 | daniel
4 | jacky
```

Richard can also choose to view the list of customers who have registered as AEMS members by selecting option `[3] Display record details`, followed by `[c] Registered Customer`. However, notice how Richard **does not have access to view their passwords**, only their username. This is an **intentional design** as AEGIS wants to protect the privacy of AEMS customer, therefore it has left their password out so **unauthorized access would not occur**. The reason their username is not capitalize is because there may be two separate accounts registering as 'Jacky' and 'jacky', hence AEGIS is designed to take that as two separate accounts rather than one.

Admin View Customer Payment

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 3
display options:
 a. Event Category
 b. All Events
 c. Registered Customers
 d. Customer Payment
Enter your option: d
here are all the payment details:
1 |
Member Name: Daniel
Event Registered: VALORANT Masters
Payment details: paid on 2022-05-28 22:48:57.335111

2 |
Member Name: Florence
Event Registered: Pet Expo 2022
Payment details: paid on 2022-05-28 22:48:57.335523

3 |
Member Name: Zorus
Event Registered: VALORANT Masters
Payment details: paid on 2022-05-28 22:48:57.335235

4 |
Member Name: Jacky
Event Registered: Taekwondo Poomsae Showcase
Payment details: paid on 2022-05-28 22:48:57.335452

5 |
Member Name: Jacky
Event Registered: Pet Expo 2022
Payment details: paid on 2022-05-28 22:48:57.335423

6 |
Member Name: Zorus
Event Registered: Education Fair KLCC
Payment details: paid on 2022-05-28 22:48:57.335324

7 |
Member Name: Daniel
Event Registered: Jay Chow Fan Meeting
Payment details: paid on 2022-05-28 22:48:57.335325

8 |
Member Name: Daniel
```

Similarly, by selecting option `[3] Display record details`, followed by `[d] Registered Customer`. Richard can view the payment details of each customer, such as the event purchased and payment date/time.

Admin Search Customer (present + not present)

1 – Username Present

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 4
Please select a search option:
 a. Search Customer
 b. Search Customer's payment
Enter your option: a
Enter customer username to search: zorus
zorus is a member of Asian Event Management Service
```

2 – Username NOT Present

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 4
Please select a search option:
 a. Search Customer
 b. Search Customer's payment
Enter your option: a
Enter customer username to search: jamesbond
jamesbond is NOT a member of Asian Event Management Service
```

If Richard chooses option `[4] Search specific record`, followed by `[a] Search customer`, he could type a username in order to initiate a search, which will return whether the username IS or IS NOT a member of Asian Event Management Service. In this example, `zorus` is a member while `jamesbond` isn't.

```
# Admin Search Customer Payment
```

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 4
Please select a search option:
 a. Search Customer
 b. Search Customer's payment
Enter your option: b
Enter customer username to search payment details: zorus

=== Here are the payment details for Zorus ===
Zorus, VALORANT Masters, paid on 2022-05-28 22:48:57.335235
Zorus, Education Fair KLCC, paid on 2022-05-28 22:48:57.335324
Zorus, Open Mic 2022, paid on 2022-05-29 00:20:11.412315
Zorus, Taekwondo Poomsae Showcase, paid on 2022-05-29 00:20:11.412315
Zorus, Jay Chow Fan Meeting, paid on 2022-05-29 01:22:07.263106
Zorus, Pet Expo 2022, paid on 2022-05-29 01:22:07.263106
```

Additionally, Richard can also search customer payments by selecting option `[4] Search specific record`, followed by `[b] Search Customer's Payment`. This will call the `adminSearchCustomerPayment()` function in which Richard will be prompted to insert a username (example: `zorus`). Then, all associating events that zorus have purchased will be displayed along with the purchase details. This allows Richard to easily see each customer's payment details simply by searching their names.

Admin Exits AEGIS

```
 What would you like to do?

1. Add event
2. Modify event records
3. Display record details
4. Search specific record
5. Exit
Your choice: 5

You have exited the application
Thank you for using our services!
```

If the admin were to select option `[5] Exit`, the `exitPage()` function will be called, which will thank the admin for using AEGIS, and exit the application.

Customer Options

```
=== Welcome to Asian Event Management Service (AEMS) ===
Please select your position:
1. Admin
2. Customer
3. Exit
Please select an option: 2

>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice:
```

When user selects option `[2] Customer` from the homepage, they will be given the role of guest, with 5 options, which are `[1] View Category`, `[2] View Event`, `[3] Log in as a Member`, `[4] Register as a Member`, and `[5] Exit`.

Customer View Category

```
>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice: 1

here are the available event categories:

1. Sports & Martial Arts
2. Musical Performance
3. Dance & Performing Arts
4. Expo & Exhibition
5. Competition & Tournament
6. Others

press <enter> to return
```

If a guest would like to view the categories that AEMS has to offer, they can select option `[1] View Category`, which will print out the 6 available categories that is provided by AEMS.

```
# Customer View Event
```

```
>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice: 2

here are all the event & their details:

1 | Taekwondo Poomsae Showcase
Category: Sports & Martial Arts
Organizer: APU Taekwondo Club
Date: 04/06/2022
Time: 13:00
Duration: 2 hours
Venue: APU Atrium
Price (RM): 10.00


2 | Battleground 2022
Category: Dance & Performing Arts
Organizer: Asia Momentum Media
Date: 22/12/2022
Time: 10:00
Duration: 3 days
Venue: DWI Emas Intersation School - Audi 2
Price (RM): 15.00


3 | Open Mic 2022
Category: Musical Performance
Organizer: Sunway Music Association
```

Alternatively, if the guest wishes to view events itself, they may select option `[2] View Event` to display all active upcoming events that are available for sign-ups.

Member Login (invalid credentials)

```
>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice: 3
>>> Member Login Page <<<
Enter your username: florence
Enter your password: hushpuppies

 >>> invalid credentials...

would you like to sign-up?
 [Y] for membership registration
 [N] to retry login
 Your choice:
```

If the customer wishes to login, they may do so by selecting the third option `[3] Log in as a Member` and insert their credentials into the respective fields. In this scenario, `Florence` have inserted the wrong credentials, but she was prompted with an option to `[Y] sign-up` or `[N] retry`, which can help guide her out of her misery.

Member Login (valid credentials)

```
>>> Member Login Page <<<
Enter your username: florence
Enter your password: ilovechatting
Login successful!
Welcome back, Florence

what would you like?
1. Add events to Shopping Cart
2. View purchased events
3. Exit
Your Choice:
```

When Florence has successfully inserted the correct credentials, she will be welcomed by AEGIS, and new features such as adding events to cart, and viewing purchased events are unlocked.

```
# Member Registration (username taken)
```

```
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username: florence
Enter a password: currykatsu
Confirm your password: currykatsu
Username already taken, please try again
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username:
```

If a guest selects option `[4] Register as a Member`, they will be brought to the member sign-up page. In this scenario, the username `florence` is already taken, hence the user is asked to retry.

```
# Member Registration (password does not match)
```

```
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username: loki
Enter a password: umaruchan
Confirm your password: umarusan
Password does not match, please try again
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username:
```

If the username is unique, but the passwords entered does not match, an error will be displayed, and the user will be required to change their credentials.

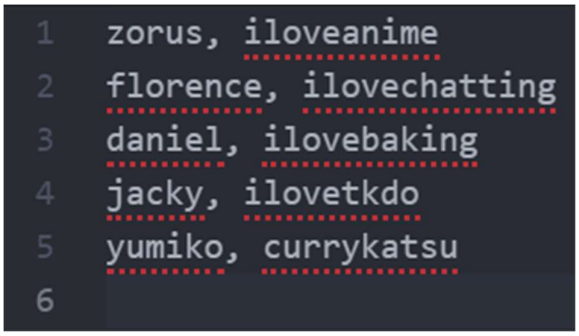
```
# Member Registration (password too short)
```

```
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username: loki
Enter a password: fight
Confirm your password: fight
Password must be longer than 8 characters, please try again
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username:
```

If the username is unique, but the password is too short, an error will be printed, and the user will be required to change their credentials.

Member Registration (successful)

```
>>> Welcome to AEMS member registration page, please follow the instruction below <<<
Enter a username: yumiko
Enter a password: currykatsu
Confirm your password: currykatsu
Registration Successful!

>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice:
```



If the guest has passes all the validation test, his/her credentials will be saved into the `memberList.txt` database, as shown above.

Member Options

```
Login successful!
Welcome back, Yumiko

what would you like?
1. Add events to Shopping Cart
2. View purchased events
3. Exit
Your Choice:
```

After successful registration and login, Yumiko can now access the members options such by selecting `[1] Add events to Shopping Cart`, `[2] View purchased events`, or `[3] Exit`.

```
# Member Add Events to Shopping Cart
```

```
what would you like?
1. Add events to Shopping Cart
2. View purchased events
3. Exit
Your Choice: 1
Which event would you like to add?
1 | Taekwondo Poomsae Showcase
2 | Battleground 2022
3 | Open Mic 2022
4 | Education Fair KLCC
5 | VALORANT Masters
6 | Jay Chow Fan Meeting
7 | Pet Expo 2022
8 | Karate Sparring Competition (Ksc)
Your choice: 8
do you want to add more items? <Y/N>Y
Which event would you like to add?
1 | Taekwondo Poomsae Showcase
2 | Battleground 2022
3 | Open Mic 2022
4 | Education Fair KLCC
5 | VALORANT Masters
6 | Jay Chow Fan Meeting
7 | Pet Expo 2022
8 | Karate Sparring Competition (Ksc)
Your choice: 1
do you want to add more items? <Y/N>N
proceeding to payment...
```

Yumiko is a martial art practitioner, and she have decided to sign-up for both the Taekwondo Poomsae Showcase and Karate Sparring Competition (KSC). Here's how she does it. First, she selects option `[1] Add events to Shopping Cart`, which will display all available events. Then she chooses `[1] Taekwondo Poomsae Showcase`, and `[Y]` to `add more items`. She then proceeds to choose `[8] Karate Sparring Competition (KSC)`, and `[N]` to proceed to payment.

```
# Member Make Payments (Insufficient amount received)
```

```
proceeding to payment...
total amount dued:  20.0
please enter your payment amount: 10
insufficient amount, please try again
total amount dued:  20.0
please enter your payment amount:
```

The first validation checks whether the payment amount is below the total amount. In this case, Yumiko only entered `(RM)10.00` into the input, which will print `"insufficient amount, please try again"` and re-loop the payment box.

```
# Member Make Payments (Invalid Datatype)
```

```
total amount dued:  20.0
please enter your payment amount: 10 peanuts

>>> please insert a valid amount!

total amount dued:  20.0
please enter your payment amount:
```

In this scenario, Yumiko tries to be cheeky and insert "10 peanuts" into the payment box. AEGIS recognized this discrepancy in datatype and print a message asking Yumiko to enter a valid amount.

```
# Member Make Payments (Exact amount or Overpay)
```

```
total amount dued:  20.0
please enter your payment amount: 50
payment successful! RM 30.0 have been returned to your account

Order Summary:
You bought: ['Karate Sparring Competition (Ksc)', 'Taekwondo Poomsae Showcase']
Total cost: 20.0
You paid: 50.0
Balance returned: 30.0
order completed! thank you for your support!
```

Finally, Yumiko entered (RM) `50.00` which will render the payment successful with a balance of (RM) `30.00` returned to Yumiko under the order summary that is printed. The order summary consists of both the events Yumiko have signed-up, the total cost, the total paid, the balance and a thank you message.

```
# Member View Purchased Events
```

```
what would you like?
1. Add events to Shopping Cart
2. View purchased events
3. Exit
Your Choice: 2

=== Here are all your registered events ===
Yumiko, Karate Sparring Competition (Ksc), paid on 2022-05-31 19:25:15.891167
Yumiko, Taekwondo Poomsae Showcase, paid on 2022-05-31 19:25:15.891167
```

After the payment have been completed, Yumiko can now check her purchase history by clicking on option `[2] View purchased events`. This option will display both the Taekwondo and Karate event that she has just bought, with the purchased date & time.

testrun.py | cartList.txt

```
1  Daniel, VALORANT Masters, paid on 2022-05-28 22:48:57.335111
2  Florence, Pet Expo 2022, paid on 2022-05-28 22:48:57.335523
3  Zorus, VALORANT Masters, paid on 2022-05-28 22:48:57.335235
4  Jacky, Taekwondo Poomsae Showcase, paid on 2022-05-28 22:48:57.335452
5  Jacky, Pet Expo 2022, paid on 2022-05-28 22:48:57.335423
6  Zorus, Education Fair KLCC, paid on 2022-05-28 22:48:57.335324
7  Daniel, Jay Chow Fan Meeting, paid on 2022-05-28 22:48:57.335325
8  Daniel, Pet Expo 2022, paid on 2022-05-28 22:48:57.335327
9  Florence, Education Fair KLCC, paid on 2022-05-28 22:48:57.335783
10 Florence, VALORANT Masters, paid on 2022-05-25 22:48:57.335231
11 Zorus, Open Mic 2022, paid on 2022-05-29 00:20:11.412315
12 Zorus, Taekwondo Poomsae Showcase, paid on 2022-05-29 00:20:11.412315
13 Zorus, Jay Chow Fan Meeting, paid on 2022-05-29 01:22:07.263106
14 Zorus, Pet Expo 2022, paid on 2022-05-29 01:22:07.263106
15 Jacky, Battlegrounds 2022, paid on 2022-05-31 12:09:24.041689
16 Jacky, Jay Chow Fan Meeting, paid on 2022-05-31 12:19:07.106904
17 Yumiko, Karate Sparring Competition (Ksc), paid on 2022-05-31 19:25:15.891167
18 Yumiko, Taekwondo Poomsae Showcase, paid on 2022-05-31 19:25:15.891167
19
```

Her details are also saved on the `cartList.txt` database, so next time she logs onto AEGIS, she can view the same purchase history that she has made weeks ago.

Member Exits AEGIS

```
what would you like?
1. Add events to Shopping Cart
2. View purchased events
3. Exit
Your Choice: 3

You have exited the application
Thank you for using our services!
```

After the purchase, Yumiko had to rush for a meeting, so she presses `[3] Exit` which will print a thank you message and close the AEGIS application.

Guest Exit AEGIS

```
>>> What would you like to do?

1. View Category
2. View Event
3. Log in as a Member
4. Register as a Member
5. Exit
Your choice: 5

You have exited the application
Thank you for using our services!
```

Similarly, a guest who's just surfing through AEMS's website can choose to exit the application anytime simply by choosing `[5] Exit`, which will print a thank you message and exit AEGIS.

Homepage exit AEGIS

```
=== Welcome to Asian Event Management Service (AEMS) ===
Please select your position:
1. Admin
2. Customer
3. Exit
Please select an option: 3

You have exited the application
Thank you for using our services!
```

Lastly, users who accidentally booted AEGIS can immediately exit at the homepage by choosing option `[3] Exit`, which will print a thank you message followed by a safe termination of AEGIS.

Conclusion

Evidently, the Asian Event General Interface Solution (AEGIS) program does contain some flaws, such as having a comparatively large file size due to the inability to use dictionaries to condense the inputs. Aside from that, the inability to use other built-in functions such as `getpass()` from the python library means that we cannot encrypt passwords into an asterisk format (example: *******) for improved security and privacy.

However, AEGIS does have a silver lining – it is a program developed with not just the users, but also the programmers in mind – built with a solid foundation, custom functions, and detailed comments within the python file, AEGIS can easily be debugged should any issues arise in the future. AEGIS have proved its ability to improve customer experience, ameliorate managements process, and increase organizing efficiency of AEMS. As such, AEGIS should be maintained, and updated regularly with new, enhanced features so that it can aid AEMS in becoming a prosperous and successful company in their future endeavours.

Workload Matrix

Name	Task	Signature
Amadea Lim Yi Wen (TP064038) UCDF2104ICT(SE)	Pseudocode Design	
	Flowchart Design	
	Proofread Documents	
	Test-run Python Codes	
	Design of the Program	
	Conclusion	
	Workload Matrix	
	Reference	
Gan Ming Liang (TP063338) UCDF2104ICT(SE)	Python code design	
	Cover Page & Formatting	
	Debugging	
	Introduction and Assumptions	
	Program Source Code with Explanation	
	Additional Features with Explanation	
	Screenshot of Input/Output with Explanation	

References

GeeksforGeeks. (28 January, 2022). *Python Try Except*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-try-except/

Programiz. (n.d.). *Python String split()*. Retrieved from Programiz: https://www.programiz.com/python-programming/methods/string/split

renderBucket. (29 December, 2020). *Learn Python By Making Programs - Shopping Cart [Part 01]*. Retrieved from YouTube: https://www.youtube.com/watch?v=j9Txy-dROyM

techWithId. (19 May, 2021). *Python Login System: Using a text file (Beginners Project)*. Retrieved from YouTube: https://www.youtube.com/watch?v=dR_cDapPWyY

w3schools. (n.d.). *Python List index() Method*. Retrieved from w3schools: https://www.w3schools.com/python/ref_list_index.asp