

AAttender Documentation

Tse Ho Nam

ICT SBA

Table of Content

Table of Content	1
Part A: Design and Implementation.....	2
I. Introduction.....	2
II. Program Initialization	5
A. File Generation	5
B. Password Settings and Encryption.....	9
C. Internal Variable Initialization	9
III. Internal Member Editor	9
IV. Module: Check-In	12
V. Module: Duty Shift	14
VI. Feature: Debug Mode	15
VII. Feature: Configurations.....	17
A. Changing Password	17
B. Toggling Program Start Up Password Authentication	18
C. Changing Check In Frequency and Time.....	18
D. Default Settings.....	20
VIII. Feature: Data Lookup.....	21
A. Data Saving.....	21
B. Manual Data Lookup Method	22
C. Internal Data Lookup Method	22
IX. Feature: Program Top Bars and Keyboard Shortcuts.....	24
Part B: Testing and Evaluation	25
I. Introduction.....	25
II. Internal Member Editor	25
A. Adding a member	25
B. Deleting a member	25
C. Saving.....	26
III. Password Hashing.....	26
IV. Program as a whole (Test #1).....	27
A. Defining Member List	27
B. Settings.....	27
C. Testing Time	27
D. Testing Environments.....	27
E. Testing Procedures	27

V. Program as a whole (Test #2).....	29
A. Defining Member List	29
B. Settings.....	29
C. Testing Time	29
D. Testing Environment	29
E. Testing Procedures	30
Part C: Conclusion and Discussion	31
I. Conclusion.....	31
II. Improvements to be made.....	31
A. Bugs in Member Editor.....	31
B. Decrease the Limit	31
C. Root Folder.....	32
D. Input Method.....	32
III. What I've Learnt.....	32
Part D: Project Management	33
I. Gantt Chart.....	33
II. Factors affected the Process of Development.....	33
Part E: Acknowledgement	33

Part A: Design and Implementation

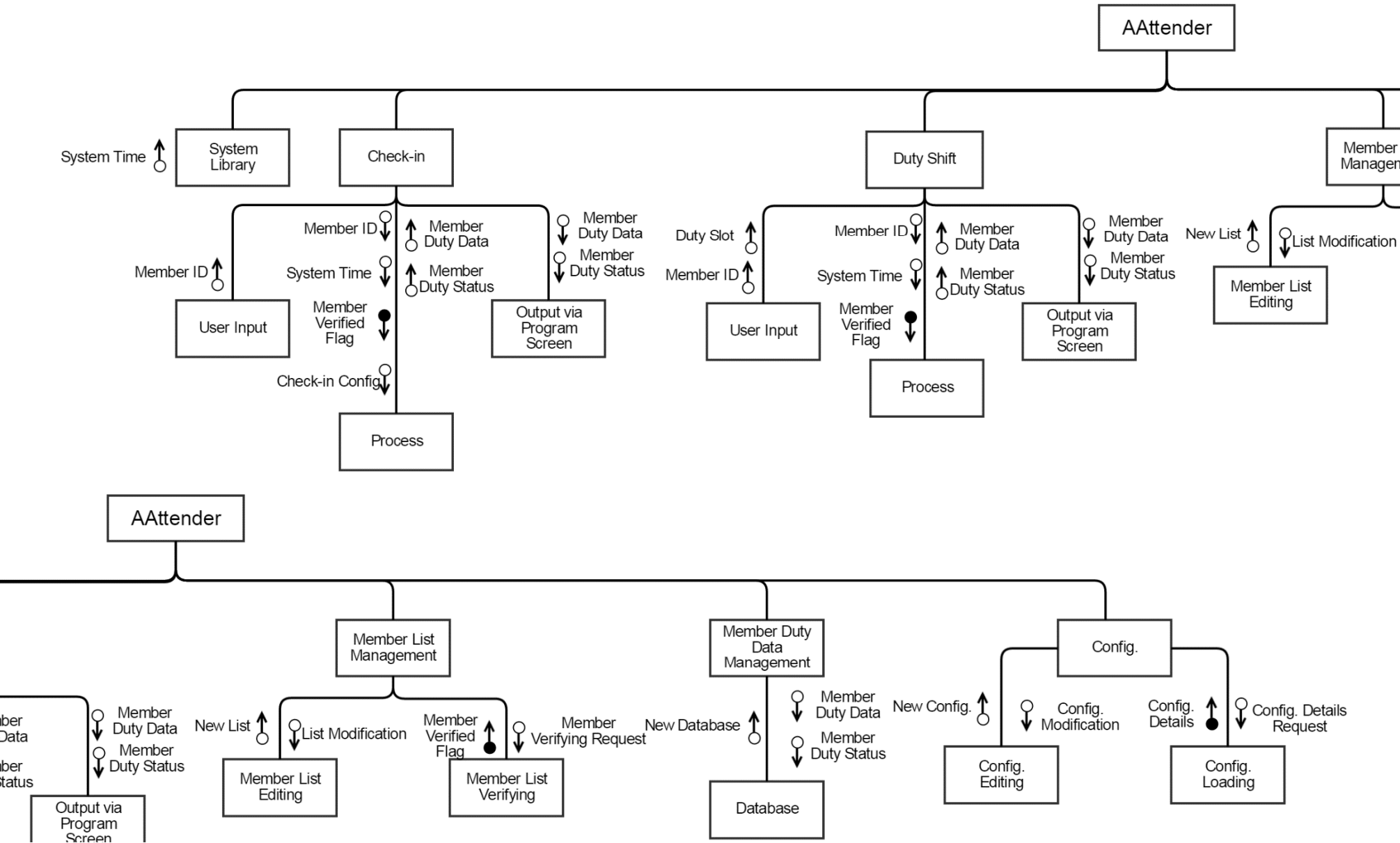
I. Introduction

AAttender is an automated attendance taking program which is designed for teams or companies that have not more than three shifts. It contains a master password for administrative controls that except the check-in module below, all functions require a password. It contains the following functions and features:

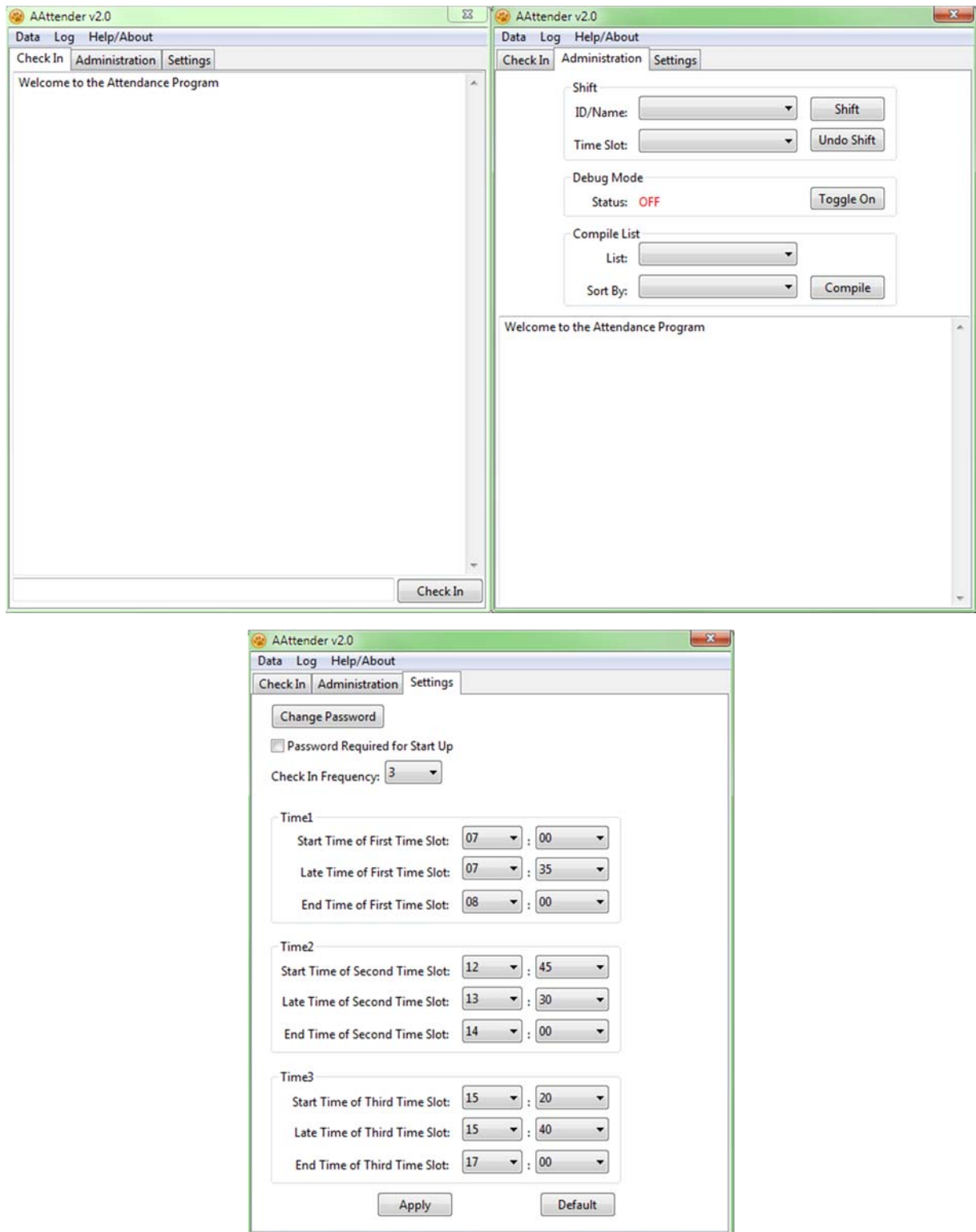
- | | |
|--------------------------|----------------------------------|
| • Internal Member Editor | • Admin Password Encryption |
| • Check-in Module | • Keyboard Shortcuts |
| • Duty Shift Module | • Program Top Bars |
| • Debug Mode | • Single Day Duty Details Lookup |
| • Configurations | • General Duty Details Lookup |

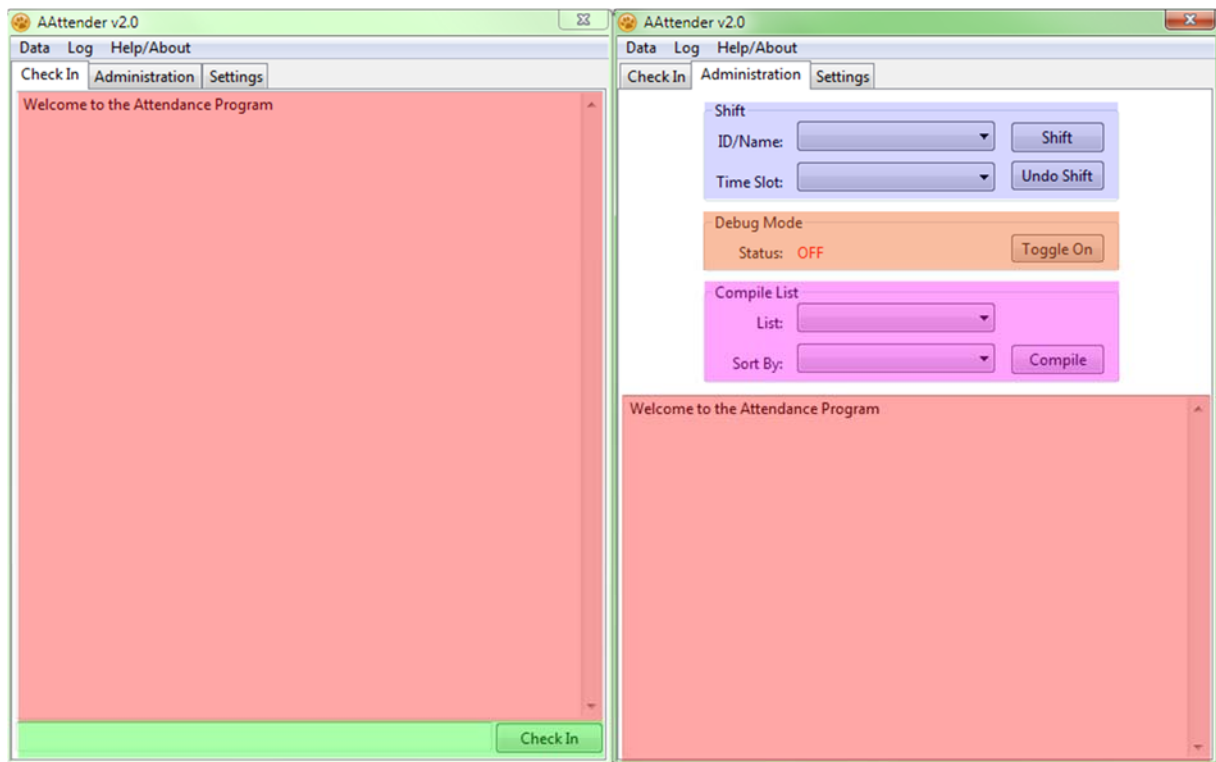
The program is designed to work on all computers with basic setup. It is not RAM or CPU intensive such that even the basic computers at school or at work can handle it.

Generally, *AAttender* has the following structure.



AAttender also has the following interface. Such graphical user interface is used because it is more user-friendly than a command line interface.





In the first tab '*Check In*', it provides the interaction of the **check-in module** to the user. The **duty shift module**, **debug mode** and **details lookup** are provided in the second tab '*Administration*' with all requiring the password. In both of the tabs, there is a **log screen** each. It is the output screen of the program that whenever there is a message from the program, it is shown on the screen. The third tab '*Settings*' controls the configurations of the program.

In the following document, each function and feature is discussed thoroughly in separate.

II. Program Initialization

A. File Generation

As all the data have to be stored, when the program is started, it checks whether `~/AAttender` exists in AAttender's current location. If the folder does not exist, the program would generating all the essential files including:

1. `~/AAttender/`

This is the root folder of all the stored information.

2. ~/AAttender/log

This is the base folder of all the log files created by the program.

3. ~/AAttender/log/dev

This is the base folder of all the warnings created by the program when Debug Mode is on and an actual conflict is detected.

4. ~/AAttender/log/YYYYMMDD.xlsx

This is a series of Excel spreadsheets containing duty details of any day that the program operates, where *YYYYMMDD* is the system date. If the program is first started, the program creates a spreadsheet with empty record and only the header. The program uses an external library *fpspreadsheet* to interact with the Excel spreadsheets with the objects provided.

This is the format of the Excel Spreadsheets:

ID	Name	Time 1	Time 2	Time 3
(ID 1)	(Name 1)	(Status 1 of Time 1)	(Status 1 of Time 2)	(Status 1 of Time 3)
(ID 2)	(Name 2)	(Status 2 of Time 1)	(Status 2 of Time 2)	(Status 2 of Time 3)
...
(ID n)	(Name n)	(Status n of Time 1)	(Status n of Time 2)	(Status n of Time 3)

5. ~/AAttender/config.ini

This is the document saving all the configurations and settings of the program with the following initialization and formatting:

```
;Configuration Files of Attendance Program
[Admin]
PW= ;The password selected by user in SHA512 hash
StartUp=0 ;1 if password is required upon start-up, 0 otherwise
```

```
[Checkin]
```

```
Freq=3 ;Range: 1 to 3, stores the number of shifts
```

```
;Stores the duty start time in Time#_Start
```

```
;Stores the duty late time in Time#_late
```

```
;Stores the duty end time in Time#_end
```

```
;The format is hh:mm:ss, ss should be 00
```

```
Time1_Start=7:00:00
```

```
Time1_Late=7:35:00
```

```
Time1_End=8:00:00
```

```
Time2_Start=12:45:00
```

```
Time2_Late=13:30:00
```

```
Time2_End=14:00:00
```

```
Time3_Start=15:20:00
```

```
Time3_Late=15:40:00
```

```
Time3_End=17:00:00
```

6. ~/AAttender/Data.xlsx

This is the document that centralized all the data and group them together. Combining with basic knowledge of Excel, the administrator may find out the show-up rate, late count by viewing the raw data containing:

- Duty Count 1, 2, 3 (DCount1, DCount2, DCount3 as seen in code)
This stores the number of duty one person has taken in duty slot 1, 2, 3
- Absent Count 1, 2, 3 (ACount1, ACount2, ACount3 as seen in code)
This stores the frequency of one person being absent in duty slot 1, 2, 3
- Shift Count 1, 2, 3 (SCount1, SCount2, SCount3 as seen in code)
This stores the number of duty one person has yet to be taken in duty slot 1, 2, 3 due to duty shift
- Late Count 1, 2, 3 (LCount1, LCount2, LCount3 as seen in code)
This stores the frequency of one person being late in duty slot 1, 2, 3
- Total Count 1, 2, 3 (TCount1, TCount2, TCount3 as seen in code)
This stores the number of duty one person should have taken in duty slot 1, 2, 3

7. ~/AAttender/members.ini

This is the file containing all the information of the members in the team or the company.

The configuration file contains the following header which states the number of members.

```
;Configuration Files of Members List
[Main]
Total=1
```

Following the header, it stores the information of the members. For the i^{th} member, it stores the details as follow:

```
[i] ;Section name is i for the i-th member
Name= ;Name of the i-th member
ID= ;ID of the i-th member
;The following stores the duty details of the i-th member
;For XXX#, where XXX is the initial of days of week and # is the
number of the duty slot, 1 means the i-th member needs to be on
duty during that specific duty slot, 0 otherwise
Mon1=
Mon2=
Mon3=
Tue1=
Tue2=
Tue3=
Wed1=
Wed2=
Wed3=
Thu1=
Thu2=
Thu3=
Fri1=
Fri2=
Fri3=
Sat1=
```

Sat2=
Sat3=
Sun1=
Sun2=
Sun3=

All the above files may auto-regenerate if corrupted or gone missing. The missing documents then would be the same as it is first started and the program will not crash.

B. Password Settings and Encryption

After all the essential files are generated, the program either requests a password or asks to create a password depending on whether it is the first time using the program. The program uses a SHA512 hash to store the password in *config.ini*. The program uses the encrypted hash to compare and check the password, as SHA512 cannot be decrypted. An external library *DCPcrypt* is used as to use the SHA512 object provided to hash the password.

Password is required in every modules except Check-in. Therefore, the password is very important and thus encrypted.

C. Internal Variable Initialization

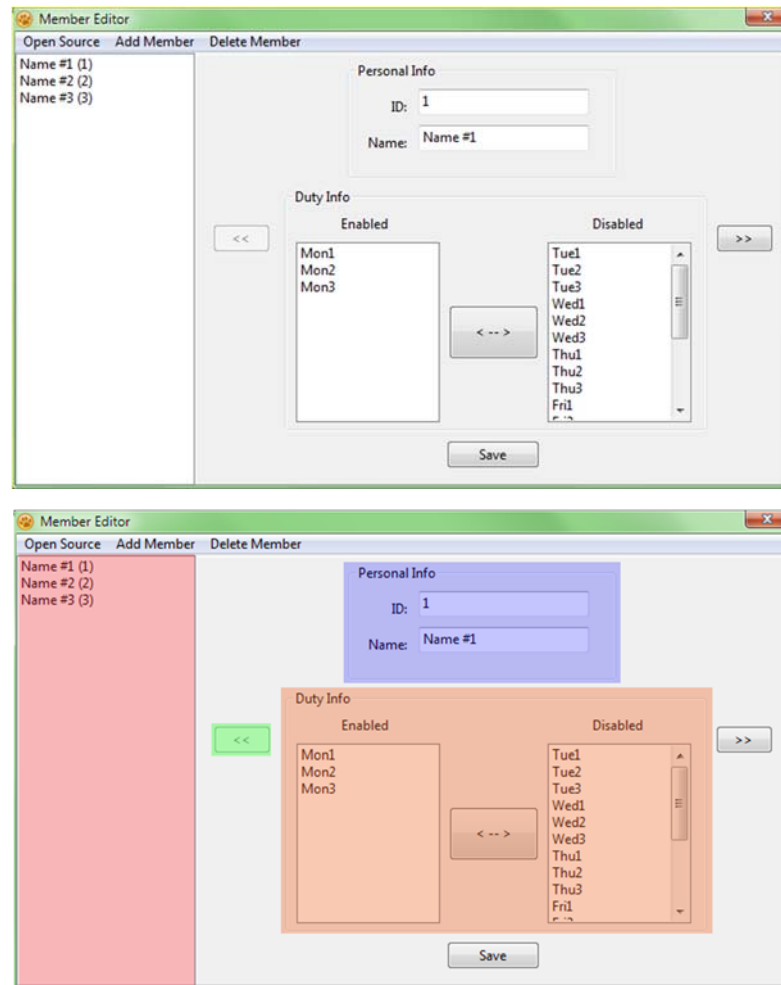
The program reads and stores the details of all members in *members.ini* to the internal storage if there are members. Otherwise, the program outputs 'WARNING: MEMEBER LIST IS MISSING' in the log screen.

Then, it reads the duty status details in */log/YYYYMMDD.xlsx* and store them in the internal storage. If it does not exist, a new Excel spreadsheet is generated with all the status preset as *Absent*. Total Count (TCount1, TCount2, TCount3) and Absent Count (ACount1, ACount2, ACount3) would also increase by 1 if one should be on duty during that slot.

III. Internal Member Editor

Technically, the administrator may edit the member list via text editor. However, if the member list is too long, it might be very complicated if one would like to edit it.

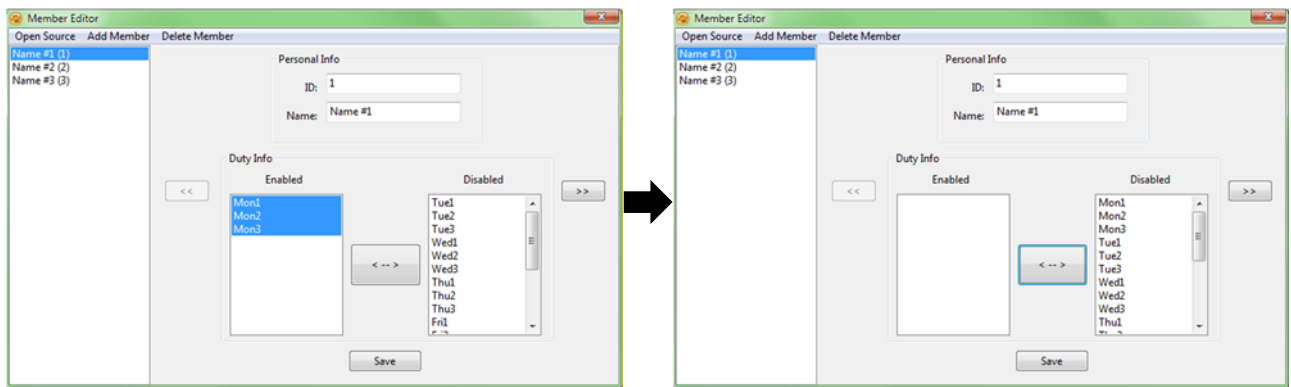
Therefore, the Internal Member Editor is created to simplify the work by making a graphical user interface for it.



The **red** zone to the left is the member list. One may double click on the name in order to edit the details. An alternative way is to click on the buttons marked with **green**. The button with '<<' can navigate to the previous member, and for the button with '>>', it navigates to the next member.

The zone marked with **blue** is the personal information. In this the administrator can change the ID and Name of the member if necessary.

For the zone coloured with **orange**, it is the duty information that stores which duty slot the member has to be on. The administrator may multi-select by pressing Shift in either the side with 'Enabled' or 'Disabled'. After selecting the slots, by pressing the button in the middle marked with '< -- >', it changes the selected slot to the other side. (Enabled to Disabled or Disabled to Enabled).



By clicking 'Open Source', after requesting the admin password, the program opens the directory to where the files are stored. This allows the administrator to check the documents without reciting the exact location of them. Admin password is also essential because the directory of the documents should be well hidden. One may find out the location of the documents if the password is not there and the data may be modified. Administrator may also create or delete an entry simply by clicking the corresponding buttons at the top. When a new member is created, it is automatically named 'Unnamed #n' and is given the ID n depending where n is the total number of members. After editing the member list, saving is crucial otherwise the changes may not be effective. The saving mechanism is done by creating a dummy *members.old* before anything is edited, and everything is automatically saved to *members.ini* if there are any actions. If the editor is closed without saving, *members.old* automatically changes back to *members.ini* and none of the changes will be saved. Otherwise, *members.old* will delete itself instead.

When a member is added and saved, a new entry is added to the */log/YYYYMMDD.xlsx* and *Data.xlsx*, where *YYYYMMDD* is the current date. The *YYYYMMDD.xlsx* would log the new member at that day with all the status 'Absent', and for *Data.xlsx*, all the values 0. If the new member is set to be on duty at the same day as the entry is added, ACount and TCount is not added to the entry as it is supposed that the member is added after the duty hour or during day offs.

```
//directory stores the root folder of AAttender
procedure TForm_memedit.FormActivate(Sender: TObject);
begin
    /* Start of Excerpt */
    CopyFile(directory + '\members.ini', directory + '\members.old');
```

```

    /* End of Excerpt */
end;

procedure TForm_memedit.btn_saveClick(Sender: TObject);
begin
    /* Start of Excerpt */
    DeleteFile(directory + '\members.old');
    CopyFile(directory + '\members.ini', directory + '\members.old');
    /* End of Excerpt */
end;

procedure TForm_memedit.FormClose(Sender: TObject; var CloseAction:
TCloseAction);
begin
    /* Start of Excerpt */
    DeleteFile(directory + '\members.ini');
    CopyFile(directory + '\members.old', directory + '\members.ini');
    DeleteFile(directory + '\members.old');
    /* End of Excerpt */
end;

```

IV. Module: Check-In

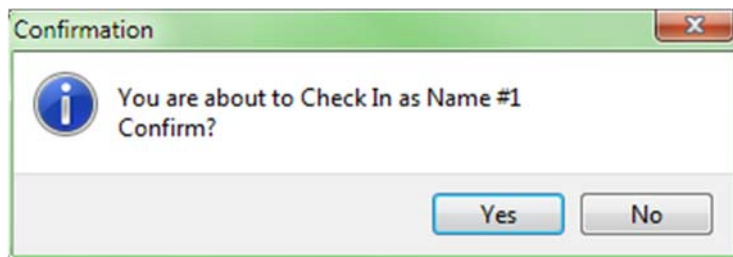
One of the essential part of the program itself is the check-in module, which aim to log the duty status of the members during the duty days. There are in total four status one member in one duty slot may get when one checks in: *OnDuty*, *Present*, *Late* with the following meanings:

Status	Meaning
OnDuty	One has checked in to an enabled duty on time
Present	One has checked in to a disabled duty
Late	One has checked in to an enabled duty late

As if the member has never checked in in one particular slot, he/she is absent regardless when it is his/her enabled duty. Therefore, another status *Absent* is assigned to all members **initially** and the new status can overwrite it when one is checked in.

In an actual operation, when one enters an ID into the text box and presses enter, there are several checks would initiate.

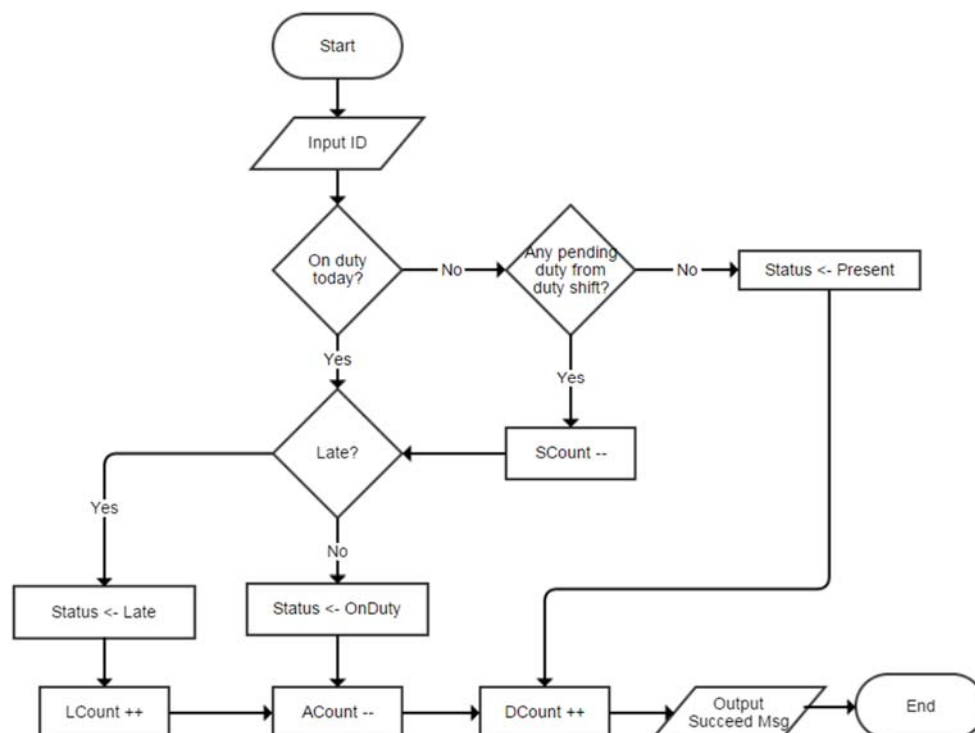
1. Check if the actual ID exists
2. Ask for confirmation via a message box



3. Check if it is time to check in
4. Check if the ID has checked in

If at least one of above tests fail, the check-in process terminates itself to prevent further errors.

Then, the program figures out which duty slot it should be when the operation is initiated by if-clauses. Following is the logical flow chart below. Technically, for different time slots, the involved variables are different (i.e. XCount1, XCount2, and XCount3). But as to be convenient, the following chart would use XCount for general cases.




V. Module: Duty Shift

Another essential module to this program is the duty shifting function. It provides a method such that the member may swap his/her duty time if he/she would like to have a day off. A password is required for this function because it would not be great if one shift the duty without any approved reason. It is also to disallow an issue of 'Unlimited Duty Shift' that one can swap every duty of his/her own and never really show up for duty.

A new status *ntdabs* may show up when this function is in use. The status *ntdabs* stands for *Noted Absent*, and will show up and replace the original *Absent* status if and only if one swap his/her duty and coincidentally he/she has a duty that exact day. This shows the administrator this very member took a day off if the administrator is looking for him/her. Note that however, *ntdabs* will not show up otherwise. The status *Absent* will still show up for his/her next duty because the program does not know which duty day the one swaps. Administrator then may use the feature (Data Lookup) to see as an alternative method.

Duty shift may also be undone if one would like to cancel it.

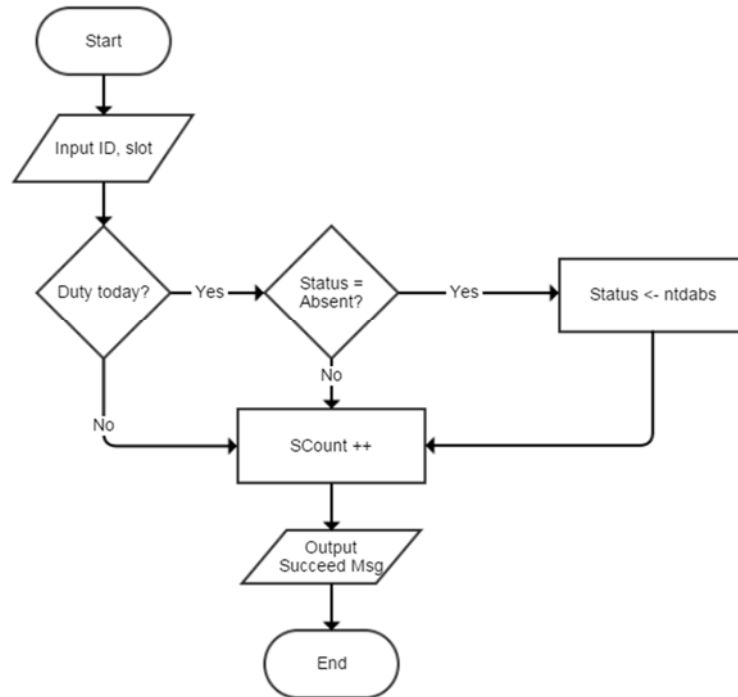


During an actual operation, the administrator should select the member in the selection box 'ID/Name' and the time slot in selection box 'Time Slot'. Then, depending on which operation one would like to do, one may click on the corresponding button. The following checks then are initiated.

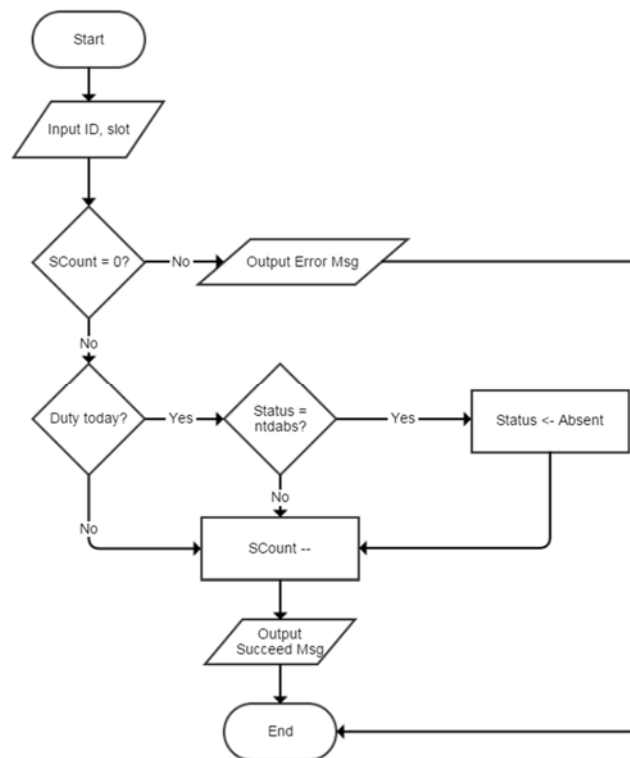
1. Check if the selection slots are non-empty
2. Check if the password entered is correct
3. Ask for confirmation

If at least one of the above checks fail, the operation terminates preventing further errors and crashes. Then, the logical code executes. Following is the logical flow chart below. Technically, for different time slots, the involved variables are different (i.e. *XCount1*, *XCount2*, and *XCount3*). But as to be convenient, the following chart would use *XCount* for general cases.

For 'Shift',



For 'Undo Shift',



VI. Feature: Debug Mode



Debug Mode is actually a developer's tool to see if there are any critical bugs. Initially the Debug Mode is **OFF**, however, when it is turned **ON**, the program would

automatically check if the counters (DCount, ACount, SCount, LCount, TCount) are reaching below 0 when an operation (i.e. Check-in and Duty Shift) is initiated. Because these counters should always be positive, having them reaching below zero means there might be severe bugs located in the program codes. The program would also output a string with format '*<name> <var name> <change>*' (for example '*Test #1 ACount2 - 1*') in log screen to show the administrators the change of variables for reference.

If there is indeed a confliction found when the program is running, this message is output by the program via the log screen: '*BUG DETECTED: <Name> has negative <var name>.*' to notify the administrators. In case of the administrators missing the warning message, a new file is also created in */log/dev* with the name '*<ID>_<var name>.log*'. For example, if the program detects the member '*Test #1*' with ID '*1*' have a negative value of *ACount2*, the program would output '*BUG DETECTED: Test #1 has negative ACount2.*', along with generating a file in */log/dev* named '*1_ACount2.log*'.

This check only applies to *Absent Counter* (ACount1, ACount2, ACount3) and *Shift Counter* (SCount1, SCount2, SCount3) because they are the only variables that may be decreased.

This mode, however, may make a false alarm shown in Part B, Testing and Debugging.

VII. Feature: Configurations

The configurations part provides modification to the programs to suit the situations of different companies and teams. All the settings are saved in config.ini with the format mentioned in previous part.

The screenshot displays a configuration window with the following elements:

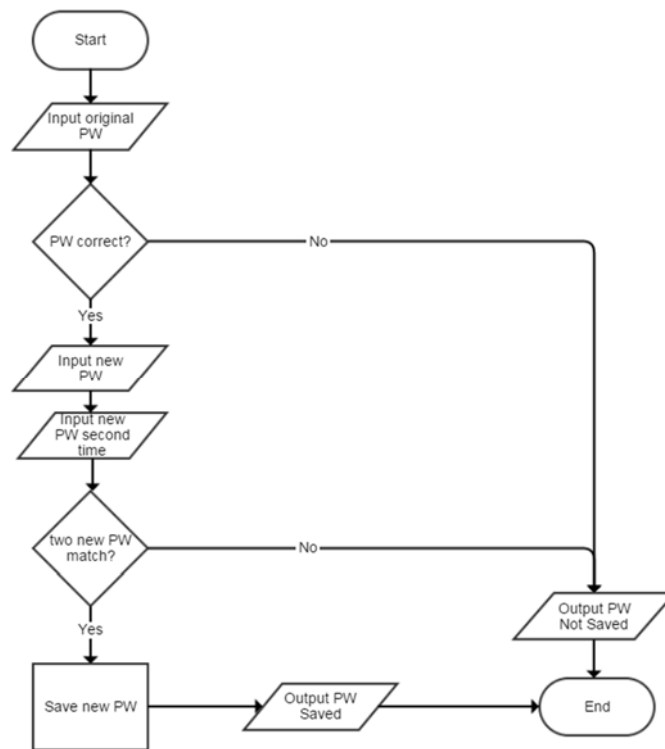
- A "Change Password" button at the top.
- A checkbox labeled "Password Required for Start Up".
- A "Check In Frequency" dropdown menu currently set to "3".
- Three time slot configuration sections: "Time1", "Time2", and "Time3". Each section contains three rows of time pickers:
 - Time1:** Start Time of First Time Slot (07:00), Late Time of First Time Slot (07:35), End Time of First Time Slot (08:00).
 - Time2:** Start Time of Second Time Slot (12:45), Late Time of Second Time Slot (13:30), End Time of Second Time Slot (14:00).
 - Time3:** Start Time of Third Time Slot (15:20), Late Time of Third Time Slot (15:40), End Time of Third Time Slot (17:00).
- "Apply" and "Default" buttons at the bottom.

In the current version, there are the following configuration settings supported:

- Changing Password
- Toggling Program Start Up Password Authentication
- Changing Check In Frequency
- Changing Start, Late and End Time of Time Slots
- Default Settings

A. Changing Password

When a user requests to change the administrator password, the program would run in the following order:



The new password, once successful changed, will be saved automatically without the need of clicking the ‘*Apply*’ button.

B. Toggling Program Start Up Password Authentication

Once enabled, the program requires the password to start. Without the password, the program terminates itself and cannot be used. When it is disabled, the program can start without requiring the password.

C. Changing Check In Frequency and Time

If the frequency is set to be below 3, at least one of the *GroupBoxes* would be disabled as the program would record the time to be all the *end time* of the latest enabled duty time. It is because the program runs in an if-clause that is

```

if time < config.citimelstart then
    /*Not the time to be on duty*/
    ...
else if time < config.citimellate then
    /*Time 1 Start*/
    ...
else if time < config.citimelend then
    /*Time 1 Late*/

```

```

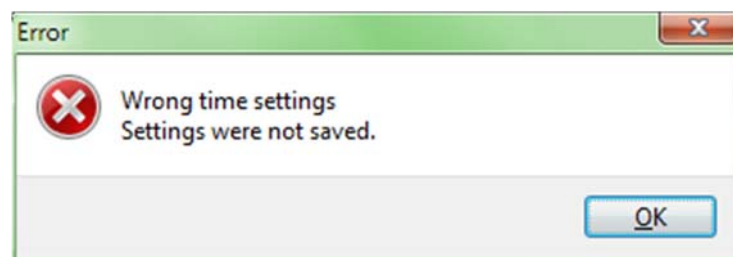
...
else if time < config.citime2start then
    /*Not the time to be on duty*/
    ...
else if time < config.citime2late then
    /*Time 2 Start*/
    ...
else if time < config.citime2end then
    /*Time 2 Late*/
    ...
else if time < config.citime3start then
    /*Not the time to be on duty*/
    ...
else if time < config.citime3late then
    /*Time 3 Start*/
    ...
else if time < config.citime3end then
    /*Time 3 Late*/
    ...
else
    /*Not the time to be on duty*/
    ...

```

If the time passes the end time of the latest enabled time slot, the following if-clauses will also be skipped, therefore, to disable the time slots, the program puts the time to be the latest enabled end time.

The start, late and end time determines the status of a member when he/she checks in, and because the time may be different in different companies and teams, it is necessary to have this support.

To prevent errors, the start time of any time slot should be earlier than the late time, and the late time of any time slot should be earlier than the end time. The time in Time 1 should all be earlier than that in Time 2, and the same for Time 2 and Time 3. If the checks fail, a warning would pop up and the settings cannot be saved.



D. Default Settings

If the administrators decide to roll back the settings to its default, by pressing the ‘Default’ button, the settings (except admin password) will be changed as it is first started.

Settings	Default
Password Required for Start Up	False

Check In Frequency	3
Time 1 Start Time	07:00
Time 1 Late Time	07:35
Time 1 End Time	08:00
Time 2 Start Time	12:45
Time 2 Late Time	13:30
Time 2 End Time	14:00
Time 3 Start Time	15:20
Time 3 Late Time	15:40
Time 3 End Time	17:00

After the settings are reset as default, it is automatically saved and the ‘*Apply*’ button is not required to be clicked for the changes to be effective.

VIII. Feature: Data Lookup

One of the most important feature that an attendance taking program should have is the ability to view all the logs easily for reference. The set of documents in */log* named *YYYYMMDD.xlsx* and the log *Data.xlsx* in root folder provides the details the administrators need when they request them. They may either use the Manual method or the program itself to view the data if they need to.

A. Data Saving

1. */log/YYYYMMDD.xlsx*

To recap, this series of Excel Spreadsheets have the format of:

ID	Name	Time 1	Time 2	Time 3
(ID 1)	(Name 1)	(Status 1 of Time 1)	(Status 1 of Time 2)	(Status 1 of Time 3)
(ID 2)	(Name 2)	(Status 2 of Time 1)	(Status 2 of Time 2)	(Status 2 of Time 3)
...
(ID <i>n</i>)	(Name <i>n</i>)	(Status <i>n</i> of Time 1)	(Status <i>n</i> of Time 2)	(Status <i>n</i> of Time 3)

The program generates one spreadsheet like this every day when the program operates, and these documents are meant to store the raw details of the status the members have in each day. This sets of spreadsheets except for storing reference details for the administrators, this is also crucial to the operations of the programs because if these documents do not exist, the program would have no way store the details of the status during the day. If the program suddenly crashes or is closed by accident, all the details are gone and may lead to crucial bugs (i.e. one may check in more than once in one time slot).

2. *Data.xlsx*

This excel spreadsheet contains all the values of the counters including Duty Counters, Absent Counters, Shift Counters, Late Counters and Total Counters which are mentioned in previous part. This is the group data that the administrators may simply view the numbers and know briefly how one is working. With simple Excel skills, the administrators may also find the attendance rate, absence rate, latency rate. The administrators may also reject duty shifts if one's Shift Counters are too high.

B. Manual Data Lookup Method

Because all the information is stored by Excel *.xlsx* format, with the appropriate spreadsheet tools such as Microsoft Excel or Apache OpenOffice Calc, the administrators may look through the details and even do some calculations.

C. Internal Data Lookup Method

The program supports basic data lookup if the administrators would like to check the data via the program itself. This provides convenience as it is no longer necessary to navigate around the system to view the data. The program provides the following data lookup only and the output would be on the log screen:

- Sum of Duty Count 1, Duty Count 2 and Duty Count 3 of a member
- Sum of Late Count 1, Late Count 2 and Late Count 3 of a member
- Sum of Absent Count 1, Absent Count 2 and Absent Count 3 of a member

- Sum of Shift Count 1, Shift Count 2 and Shift Count 3 of a member
- Sum of Total Count 1, Total Count 2 and Total Count 3 of a member
- The list of all *Present* members in current duty slot
- The list of all *Late* members in current duty slot
- The list of *Absent* members who should be on duty in current duty slot
- The list of all *Absent* members in current duty slot
- The list of all members

The program also provides a sorting method, which can be sorted by:

- Member ID
- Member Name
- Value of Counters *or* Statuses

This is an example of the outputs:

1. Sort by ID

```

ID, Name, Total Count
1, Name #1, 0
2, Name #2, 0
3, Name #3, 0
4, Name #4, 0
5, Name #5, 0

ID, Name, Status
1, Name #1, absent
2, Name #2, absent
3, Name #3, absent
4, Name #4, absent
5, Name #5, absent

```

2. Sort by Name

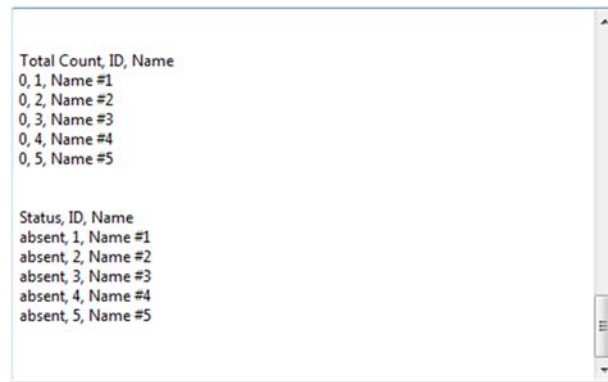
```

Name, ID, Total Count
Name #1, 1, 0
Name #2, 2, 0
Name #3, 3, 0
Name #4, 4, 0
Name #5, 5, 0

Name, ID, Status
Name #1, 1, absent
Name #2, 2, absent
Name #3, 3, absent
Name #4, 4, absent
Name #5, 5, absent

```


3. Sort by Counters/Statuses



IX. Feature: Program Top Bars and Keyboard Shortcuts

This program comes with keyboard shortcuts that user may use for convenience.

Main Program (Top Bar):

Operation	Keyboard Shortcuts	Usage
'Data' -> 'Open Directory'	Ctrl+O	Open root folder
'Data' -> 'Edit Member'	Ctrl+M	Open Internal Member Editor
'Data' -> 'Reload All'	Ctrl+R	Reload the program as if it restarted
'Log' -> 'Clear Log'	Shift+Ctrl+C	Clear the log screen
'Log' -> 'Save Log'	Ctrl+S	Save the log screen with text format
'Help/About' -> 'Help'	Ctrl+H	Provides help (not yet completed)
'Help/About' -> 'About'	Shift+Ctrl+A	Provides the program basic information
'Help/About' -> 'Dev. Log'	/	Provides the place for developer to store bug lists and to-be-developed contents lists, should be disabled when released

Internal Member Editor (Top Bar):

Operation	Keyboard Shortcuts	Usage
'Open Source'	Ctrl+O	Open members.ini
'Add Member'	Ctrl+Add	Add a member
'Delete Member'	Del	Delete a member

Part B: Testing and Evaluation

I. Introduction

Below shows the test cases for the program to determine if it is working properly. The Internal Member Editor and Admin Password Encryption will be tested separately, whereas the Check-in Module, Duty Shift Module, Debug Mode and Data Lookup will be tested as a whole with some predefined scenarios. The others are not tested because there exists no test cases available, as their tasks are simple.

II. Internal Member Editor

A. Adding a member

Name	ID	Duty Slot	Status	Note
Alex	1	Mon1, Mon2, Mon3	Pass	/
Name #1	2	All	Pass	Name is in <i>string</i> , so numbers and symbols are supported
John	A#3	Tue1, Tue2, Tue3	Pass	ID is in <i>string</i> , so texts are supported
	4	Wed1, Thu1, Fri1	Pass	There exists no test just yet for empty name disallowance, this should not pass
Amy		Sat1	Pass	There exists no test just yet for empty ID disallowance, this should not pass
Jim	4	Sun1	Pass	There exists no test just yet for duplicate ID disallowance, this should not pass
Nathan	6		Pass	This should pass in case of a disabled member

B. Deleting a member

Scenario	Status	Note
Delete when there are at least 3 members	Pass	/
Delete when there are 2 members	Pass	The '>>' is also disabled

Delete when there are 1 member	Pass	The ' <i>Delete Member</i> ' is also disabled
Delete when there are no member	Pass	The ' <i>Delete Member</i> ' is already disabled and cannot be clicked

C. Saving

Scenario	Status	Note
Save without doing anything	Pass	/
Quit without doing anything	Pass	Quit means quit without saving
Save after adding a member	Pass	/
Quit after adding a member	Pass	Quit means quit without saving
Save after deleting a member	Pass	/
Quit after deleting a member	Pass	Quit means quit without saving

III. Password Hashing

This is the test testing the SHA512 algorithm in the program. The online SHA512 generator website is used to check.

Raw	Hash	Status
123456	BA3253876AED6BC22D4A6FF53D8406C6 AD864195ED144AB5C87621B6C233B548 BAEAE6956DF346EC8C17F5EA10F35EE3 CBC514797ED7DDD3145464E2A0BAB413	Pass
AbCdEf	FEF579685DC4704EABFB0F225C72A089 CA40C022C0B882BAED72D74F6199BC54 8669263C33835881CC67C777FB52EC14 0BAD26C562D30AEA51E0BBE5E6B69A19	Pass
!4@5gdsDGb	A48511EE862C010898E11836BE86304C 592E57D53821B2C1D2F90BF9262B2DCC F570C310924DBD5D13E789CCCBF5864C 8ABD31F6E018FB035CE38050983BD8A0	Pass
	/	Pass (Empty PW not allowed)

IV. Program as a whole (Test #1)

A. Defining Member List

During the test, the following will be the members listed in the program.

Name	ID	Duty Slot
Name #1	1	Mon1, Mon2, Mon3
Name #2	2	Tue1, Tue2, Tue3
Name #3	3	Thu1, Thu2, Thu3
Name #4	4	Thu1, Thu2, Thu3
Name #5	5	Thu1, Thu2, Thu3
Name #6	6	Thu1, Thu2, Thu3

B. Settings

The Default settings with check in frequency being 1 will be used. Debug Mode is also ON.

C. Testing Time

The test is run with system time on a Thursday.

D. Testing Environments

The test is carried with a brand new *AAttender* that has not generated any files.

E. Testing Procedures

Procedure	Status	Note
Checkin with ID 1 at 07:30	Pass	DCount1 ++ Status \leftarrow <i>Present</i>
Checkin with ID 4 at 07:30	Pass	DCount1 ++, ACount1 -- Status \leftarrow <i>OnDuty</i>
Shift Duty Slot 1 with ID 5 at 07:32	Pass	SCount1 ++ Status \leftarrow <i>ntdabs</i>
Shift Duty Slot 1 with ID 2 at 07:37	Pass	SCount1 ++
Undo Duty Shift Slot 1 with ID 5 at 07:38	Pass	SCount1 –

		Status \leftarrow <i>Absent</i>
Shift Duty Slot 1 with ID 2 at 07:37	Pass	SCount1 ++
Undo Duty Shift Slot 1 with ID 2 at 07:37	Pass	SCount1 --
Checkin with ID 2 at 07:32	Pass with side notes	<p>Scount1 -- ACount1 -- DCount1 ++</p> <p>In this case ACount reaches -1, but it is yet to be increased when ID 2's duty comes. This is the only false alarm the Debug Mode may make.</p>
Checkin with ID 5 at 07:38	Pass	<p>LCount1 ++ ACount1 -- DCount1 ++ Status \leftarrow <i>Late</i></p>
Shift Duty Slot 1 with ID 6 at 07:38	Pass	<p>SCount1 ++ Status \leftarrow <i>ntdabs</i></p>
Checkin with ID 6 at 07:39	Pass with side notes	<p>LCount1 ++ ACount1 -- DCount1 ++ Status \leftarrow <i>Late</i></p> <p>Note that SCount1 does not decrease, it is because the program does not allow the duty shifted to be executed at the same slot as your original duty. Otherwise, a shifted duty may delete itself when a person carrying a shifted duty record checks in to his/her duty.</p>

V. Program as a whole (Test #2)

A. Defining Member List

During the test, the following will be the members listed in the program.

Name	ID	Duty Slot
Name #1	1	Mon1, Mon2, Mon3
Name #2	2	Tue1, Tue2, Tue3
Name #3	3	Thu1, Thu2, Thu3
Name #4	4	Thu1, Thu2, Thu3
Name #5	5	Thu1, Thu2, Thu3
Name #6	6	Thu1, Thu2, Thu3

B. Settings

The Default settings will be used. Debug Mode is also ON.

C. Testing Time

The test is run with system time on a Thursday.

D. Testing Environment

The test is carried out with *AAttender* that there are already stored data.

Here is the data before the program starts:

ID	DCount1	ACount1	SCount1	LCount1	TCount1
1	10	0	0	0	10
2	8	2	1	0	10
3	9	1	0	1	10
4	10	0	0	0	10
5	10	0	0	1	10
6	10	0	0	1	10

E. Testing Procedures

Procedure	Status	Note
Checkin with ID 1 at 07:30	Pass	DCount1 ++ Status \leftarrow <i>Present</i>
Checkin with ID 4 at 07:30	Pass	DCount1 ++, ACount1 -- Status \leftarrow <i>OnDuty</i>
Shift Duty Slot 1 with ID 5 at 07:32	Pass	SCount1 ++ Status \leftarrow <i>ntdabs</i>
Shift Duty Slot 1 with ID 2 at 07:32	Pass	SCount1 ++
Undo Duty Shift Slot 1 with ID 5 at 07:32	Pass	SCount1 -- Status \leftarrow <i>Absent</i>
Shift Duty Slot 1 with ID 2 at 07:32	Pass	SCount1 ++
Undo Duty Shift Slot 1 with ID 2 at 07:32	Pass	SCount1 --
Checkin with ID 2 at 07:32	Pass	SCount1 -- ACount1 -- DCount1 ++
Checkin with ID 5 at 07:38	Pass	LCount1 ++ ACount1 -- DCount1 ++ Status \leftarrow <i>Late</i>
Shift Duty Slot 1 with ID 6 at 07:38	Pass	SCount1 ++ Status \leftarrow <i>ntdabs</i>
Checkin with ID 6 at 07:39	Pass with side notes mentioned above	LCount1 ++ ACount1 -- DCount1 ++ Status \leftarrow <i>Late</i>

Test cases with Duty Slot 2 and 3 are also conducted with similar results. Those will not show here due to the redundancy.

Part C: Conclusion and Discussion

I. Conclusion

To review, the scenario was to design a solution for attendance taking automation. I believe the program designed (*AAttender*) is a fit solution to the program, yet there are still a bug or two in the code and some functionalities of the program are still under-developed. However, as the critical modules of the program have gone pass the test, it can be concluded that the program is generally completed. Some tweaks can be applied afterwards as to enhance the user's experiences.

II. Improvements to be made

Even though the program is generally completed, as mentioned above, some tweaks can still be applied.

A. Bugs in Member Editor

In the testing procedures, about adding a new member to the system via the internal editor, there are some bugs are yet to be fixed.

Name	ID	Duty Slot	Status	Note
	4	Wed1, Thu1, Fri1	Pass	There exists no test just yet for empty name disallowance, this should not pass
Amy		Sat1	Pass	There exists no test just yet for empty ID disallowance, this should not pass
Jim	4	Sun1	Pass	There exists no test just yet for duplicate ID disallowance, this should not pass

The following checks should be added to the editor:

1. Field Presence Check
2. Duplication Check

With these checks, the above problems should be able to be fixed.

B. Decrease the Limit

Right now the program only support up to 3 duty slots. This does not support those companies that have more than 3 duty slots, and those with different duty slot for different individuals, and even those with overnight duties. The program in the future may develop in this way, such to allow more companies to be able to use the program.

C. Root Folder

The root folder now cannot be moved, and is predetermined to be generated at where the executable file is located at. It is currently impossible to move the root folder, and moving it might be necessary because administrators would not want their colleagues to change the data at any time if the root folder is public to everyone. This should be an option in the future updates.

D. Input Method

The program is now designed to accept keyboard inputs only, yet later if possible, it should be able to accept barcode inputs. For this to work the program doesn't need to be modified, but adding the barcode scanner could be one of the external modification made to the whole system.

III. What I've Learnt

It is a great pleasure that I am able to work on a project like this. I have learnt how to write a proper GUI program, as well as the basic concepts of Object-Oriented Programming as I have to use some objects within the design. Doing the project from head to tail is a brand new experience to me, and I am very happy that I enjoyed the process.

Part D: Project Management

I. Gantt Chart

The Gantt Chart of this very project is as follow:

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
Data Structure Design								
GUI Design								
Flowchart Design								
Implementation								
System Testing								
Bug Fixing								

II. Factors affected the Process of Development

As at first I did not assume I would use *Excel* and *.ini* files as my storage, during the Implementation I have to use a bit of time to learn the concepts behind the *packages* I found online that allow me to do this. Thus the time used in Implementation was longer than expected. Similar problem happened during the GUI design, but luckily the GUI design part is easier to learn comparing to the *packages*, the time delayed in that part is not significant.

Part E: Acknowledgement

Here I would like to thanks the following sources for assisting in this project.

1. Package: *DCPCrypt* by David Barton, with barko and Graeme Geldenhuys as contributors
2. Package: *fpspreadsheet* by the community of *Lazarus*
3. IDE: *Lazarus* founded by Cliff Baeseman, Shane Miller and Michael A. Hess
4. Wiki: *Lazarus and Free Pascal Wiki* by the community of *Lazarus* and *Free Pascal*
5. Book: *NSS Information and Communication Technology* by Pearson
6. Other documents that helped in this project

[End of Report]