
# Foundation in Science

# FHCT1022: Programming Concepts

## Group Assignment

---

**Lecturer:**
**Puan.B A Jesica Malani Binti B A Sugathapala**

**Practical Group: 10**

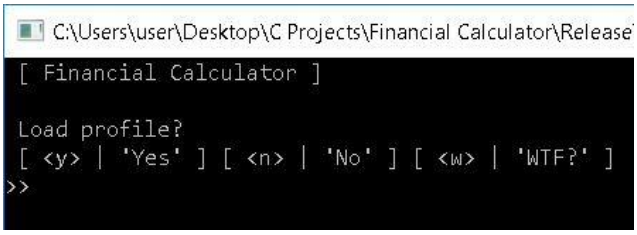

| Name | ID |
|---|---|
| Aaron Tan Pei Zian | 1800779 |
| Goh Keng Hui | 1800417 |
| Lua Pei Yuan | 1800400 |
| Fong Chien Yoong | 1800565 |

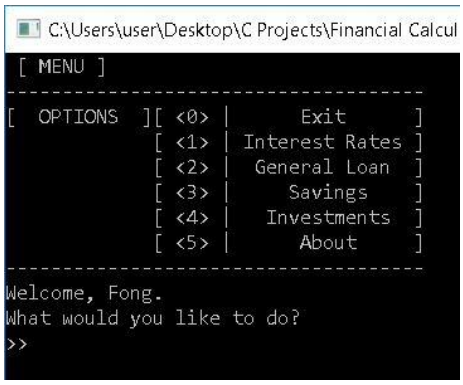
# Table of Contents

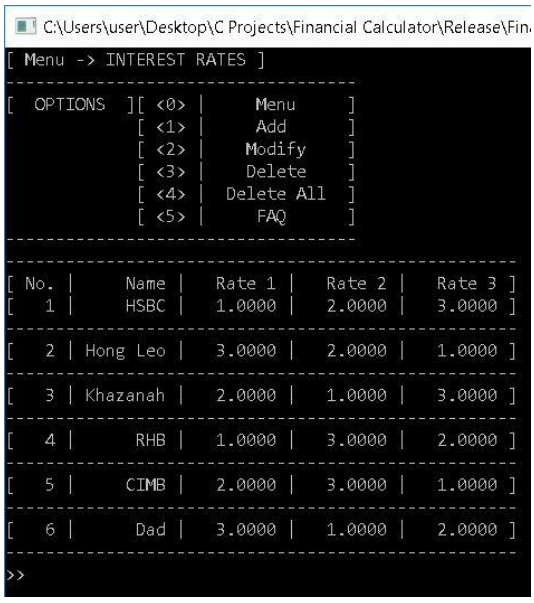
# About

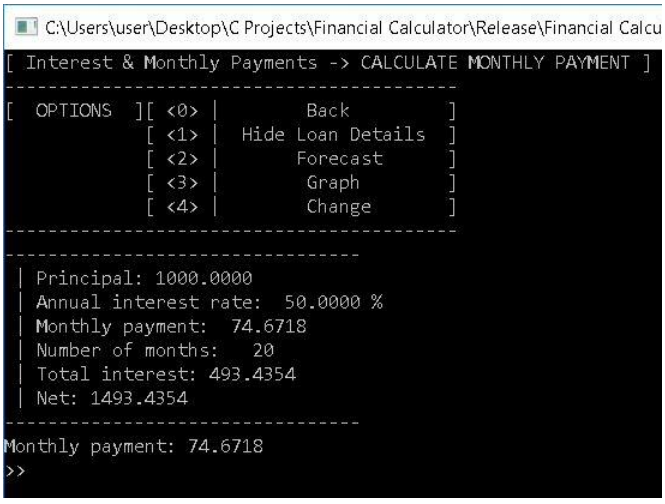
- **Program Name:** Financial Calculator
- **Programming Language:** The Infamous C, the one which gives programmers the Big C.
- **Purpose:** To pass our current semester.
- **Credits:**
    - UTAR for delaying our road to dementia.
    - UTAR lecturers for their guidance, especially Puan.B A Jesica Malani Binti B A Sugathapala, our practical lecturer.
    - varielle from SourceForge for the TinyFileDialogs library.
    - DaveGamble from Github for the cJSON library.
    - God, for keeping us alive.
- **Team:**
    - Aaron, the human.
    - Fong, the male.
    - Gary, the individual.
    - Pei Yuan, the man.
- **Team motto:** “Mamak mou?”

# Screenshots



C:\Users\user\Desktop\C Projects\Financial Calculator\Release\Financial Calculator.exe

```
[ Interest & Monthly Payments -> CALCULATE MONTHLY PAYMENT ]
---------------------------------
 | Principal: 1000.0000
 | Annual interest rate:  50.0000 %
 | Monthly payment: 124.3160
 | Number of months:   10
 | Total interest: 243.1600
 | Net: 1243.1600
---------------------------------

-----------------------------------------------------------------------------------------------------------
[ Month | Monthly Payment | Principal Part | Interest Part | Cumulative Interest |      Balance |          Net ]
-----------------------------------------------------------------------------------------------------------
[     1 |        124.3160 |        82.6493 |       41.6667 |             41.6667 |     917.3507 |     124.3160 ]
-----------------------------------------------------------------------------------------------------------
[     2 |        124.3160 |        86.0931 |       38.2229 |             79.8896 |     831.2576 |     248.6320 ]
-----------------------------------------------------------------------------------------------------------
[     3 |        124.3160 |        89.6803 |       34.6357 |            114.5253 |     741.5773 |     372.9480 ]
-----------------------------------------------------------------------------------------------------------
[     4 |        124.3160 |        93.4169 |       30.8991 |            145.4244 |     648.1604 |     497.2640 ]
-----------------------------------------------------------------------------------------------------------
[     5 |        124.3160 |        97.3093 |       27.0067 |            172.4311 |     550.8511 |     621.5800 ]
-----------------------------------------------------------------------------------------------------------
[     6 |        124.3160 |       101.3639 |       22.9521 |            195.3832 |     449.4872 |     745.8960 ]
-----------------------------------------------------------------------------------------------------------
[     7 |        124.3160 |       105.5874 |       18.7286 |            214.1118 |     343.8998 |     870.2120 ]
-----------------------------------------------------------------------------------------------------------
[     8 |        124.3160 |       109.9868 |       14.3292 |            228.4410 |     233.9130 |     994.5280 ]
-----------------------------------------------------------------------------------------------------------
[     9 |        124.3160 |       114.5696 |        9.7464 |            238.1874 |     119.3434 |    1118.8440 ]
-----------------------------------------------------------------------------------------------------------
[    10 |        124.3160 |       119.3434 |        4.9726 |            243.1600 |       0.0000 |    1243.1600 ]
-----------------------------------------------------------------------------------------------------------
Press any key to continue . . .
```

C:\Users\user\Desktop\C Projects\Financial Calculator\Release\Financial Calculator.exe

```
[ Interest & Monthly Payments -> CALCULATE MONTHLY PAYMENT ]
---------------------------------
 | Principal: 1000.0000
 | Annual interest rate:  50.0000 %
 | Monthly payment: 124.3160
 | Number of months:   10
 | Total interest: 243.1600
 | Net: 1243.1600
---------------------------------

    Balance
          ^
   1000.0--*****
          -
          -     *****
          -
          -          *****
    800.0--
          -               *****
          -
          -                    *****
    600.0--
          -                         *****
          -
          -
          -                               *****
    400.0--
          -                                    *****
          -
          -
          -                                         *****
    200.0--
          -
          -                                              *****
          -
          -
      0.0--|----|----|----|----|----|----|----|----|----|----|> Months
           0    01   02   03   04   05   06   07   08   09   10
Press any key to continue . . .
```

C:\Users\user\Desktop\C Projects\Financial Calculator\Release\Financial Calculator.exe

```
[ Interest & Monthly Payments -> CALCULATE MONTHLY PAYMENT ]
---------------------------------
 | Principal: 1000.0000
 | Annual interest rate:  50.0000 %
 | Monthly payment:  59.0060
 | Number of months:   30
 | Total interest: 770.1813
 | Net: 1770.1813
---------------------------------

    Balance
          ^
   1000.0--****
          -    ***
          -       ***
          -          ****
          -             *
    800.0--             ****
          -                *
          -                 ****
          -                    *
          -                     **
    600.0--                     ***
          -                        **
          -                         **
          -                          *
          -                          **
    400.0--                           **
          -                            *
          -                            **
          -                             **
          -
    200.0--                              *
          -                              **
          -                                **
          -
          -                                 *
      0.0--|----|----|----|----|----|----|----|----|----|----|> Months
           0    03   06   09   12   15   18   21   24   27   30
Press any key to continue . . .
```

# Preface

This program comprises of seven calculators which address varying degrees of financial computations. To simplify things, these calculators are further classified into three categories:

- General loans
- Savings
- Investments

Each of these categories is isolated into header and C files to improve the program's modularity. Other modular components also exist, with functions ranging from custom display outputs to file processing. Moreover, two external libraries were acquired, namely "cJSON" from Github and "tinyfiledialogs" from SourceForge. The former is utilised for data storage and access whereas the latter enables users to select files, a key feature of the program. Data is encoded in JSON due to its convenience and accessibility.

The algorithms employed throughout the program vary in complexity. In fact, some of them serve as workarounds for certain technicalities unique to the C dialect which unnecessarily increases the complexities of the overall program. Hence, some implementations may not be applicable to other languages, some of which are able to obtain the desired output with less effort.

It helps to comprehend the general layout the program before diving into its intricacies. Basically, the program can be divided into two subsets, calculators and interest rates. Calculators are merely sections which compute the desired output and nothing more. On the other hand, the interest rates are a unique compartment whereby the user is able to store multiple interest rates in the form of a file. The file is essentially just a text file with a name specified by the user. No specific extension is enforced, as it exerts zero influence on the file operations used.

The user's interest rates can be accessed can be accessed by any of the calculators which requires an interest rate. A consequence of implementing such a profile-based framework is that it becomes mandatory for the user to create a profile before being able to utilise the program. Due to laziness, we have not introduced a "guest mode" yet.

Some calculators, especially those which involve financial growth or loans in particular, are able to display a table of values and also a graph describing the gradual growth.

FAQs or "Frequently Asked Questions" are littered throughout the program. These sections contain general info on a particular topic, including links to PDFs which discusses the mathematical background of the topic.

For not-so-obvious reasons, a couple of data structures beyond the scope of the course are used, including structs, pointers, function pointers, linked lists and enumerators. The only bitwise operator used are '|' and '&'.

Due to pure laziness, support for Linux and Mac systems has been dropped so the program only runs on Windows.

# Part 1: Preliminaries

## 1.1: File Manager

The standard I/O library not only contains the console printing and input capabilities, but also extends them into the realm of filesystems, whether on Windows or Unix-based systems. File Manager is a component under the header “FileManager.h” which handles the file operations of the program. For convenience, we shall refer to it as FM. It constitutes of two functions which handle file reading and writing respectively.

a) The reading function is declared as follows:

```
char* fManager_ReadFile(FILE* file, bool closeStream)
```

The function receives an input stream and a boolean as inputs. The fgets function from the standard I/O library reads a line of text from an input stream. It is called repeatedly until the end of the input stream is reached. Each line obtained from fgets is concatenated to a temporary char array which will be finally returned as a char pointer. The array is arbitrarily defined to contain 3000 elements for no particular reason. The input stream will be closed if the boolean is true.

b) The writing function is declared as follows:

```
void fManager_WriteFile(FILE* file, char content[], bool closeStream)
```

The function receives an output stream, a boolean, and a text in the form of a char array which will be written to the output stream. The fprintf function from the standard I/O library executes the writing with minimal effort from us. The output stream will be closed if the boolean is true.

We think that flowcharts and pseudocode diagrams are inapplicable here. We hope that the lecturer would concur.

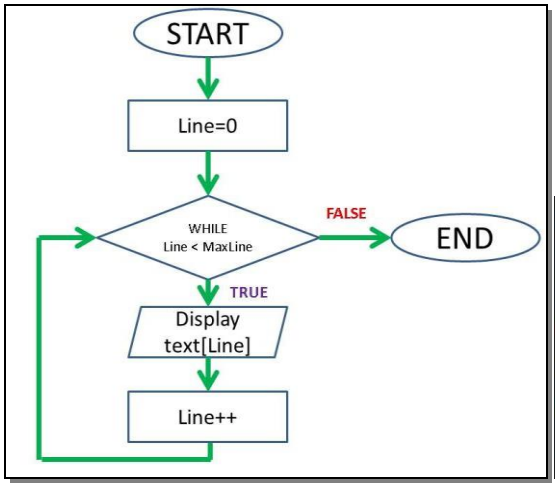
# 1.2: Custom Printf

Certain sections of the program involve a myriad of display patterns, some of which are tedious to replicate. To simplify things, they have been condensed into modular functions.

a) **Periodic Printf**

```
void PeriodicPrintf(int interval, char text[])
```

➢ Used to display texts in an aesthetically-pleasing manner, this function accesses the individual elements of a char array and displays them individually at a fixed time interval. The Windows' Sleep function is used to simulate the time intervals.

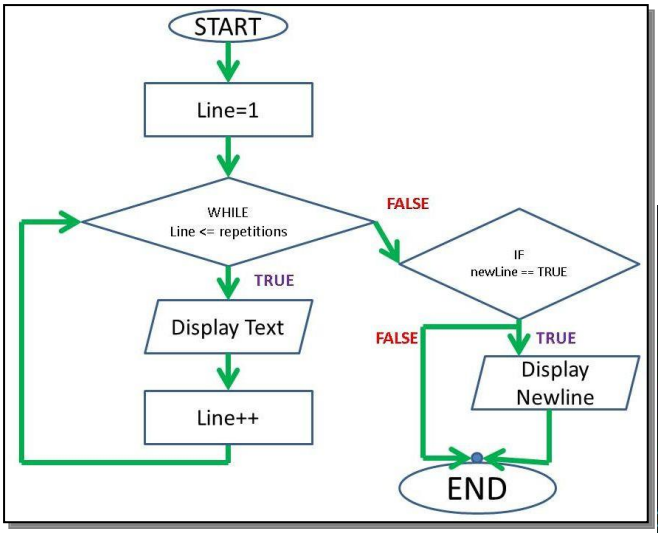



```
Periodic_Printf
    line = 0
    REPEAT
        Display text[line]
        line++
    UNTIL line >= maxLine
END
```

b) **Print Loop**

```
void PrintLoop(char text[], int repetitions, bool newLine)
```

➢ Prints a text multiple times. It will print a newline character at the end if the input boolean is true.




```
Print_Loop
    line = 1
    REPEAT
        Display text
        line++
    UNTIL line > maxLine
    IF repetitions == TRUE
        Display '\n'
    ENDIF
END
```

**c) Print Invalid Input**

```
void PrintInvalidInput()
```

➢ Prints a message describing an invalid input.

**d) Print Invalid Negative**

```
void PrintInvalidNegative()
```

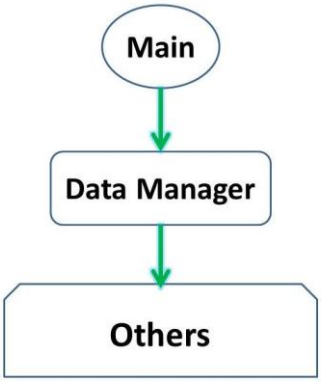
➢ Prints a message asking for a positive value.

---

The default time interval for the Periodic Printf function is specified in this header. It is 25 milliseconds for some reason.

## 1.3: Data Manager

Before proceeding to comprehend the entire program, one must inspect the component which unifies the rest. For convenience's sake, Data Manager would be represented throughout the guide by the abbreviation, DM. The following chart visually describes the role of DM:




DM is the header which links all the other headers together, allowing them to interact seamlessly. Most of the default values are defined here. The following are some of the important definitions:

a) **FLUSH**

```
char flushChar;
#define FLUSH while ((flushChar = getchar()) != '\n' && flushChar != EOF)
```

➢ Clears the input buffer. Input functions like getchar(), getch(), and scanf() share one common issue whereby a newline character is stored in the input buffer after the user presses the "Return" key. The issue emerges in scanf when it is instructed to search for characters. However, it is not apparent when scanf is only searching for digits as it will discard the newline character automatically. Apparently, the solution suggested by the lecturers involves the calling the function `fflush(stdin)`. We, however, do not concur and resorted to a viable alternative (Why?). In our solution, getchar() is called repeatedly in a loop as long as the char obtained from the input buffer is not the newline char or the EOF. Hence, the loop quits once it extracts the newline from the buffer, allowing the program to resume.

b) **PAUSE**

```
#define PAUSE system("pause")
```

➢ Pauses the program until the user enters an input.

c) **CLEAR**

```
#define CLEAR system("cls")
```

➢ Clears the screen.

The following are some of data structures unique to the DM:

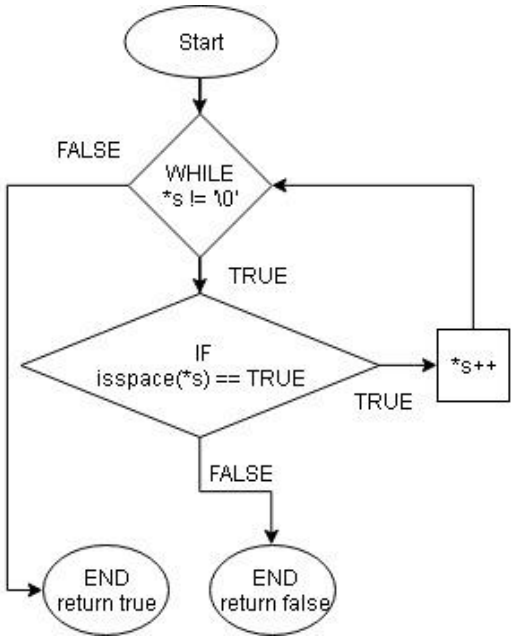
a) **FAQTypes**
   An enumerator containing unique values representing each calculator respectively.
b) **User Profile**
   A simple struct containing two fields, the user's name and his/her age.
c) **Date**
   A simple struct with two integers, the month and the year.
d) **Duration**
   To represent a duration with either dates or just months/years, this struct contains variables to accommodate those possible representations. For instance, the start and end dates will be used if the duration is representing the duration as an interval between two dates. The dates are variables with the type Date mentioned earlier in (b). Some of the fields/variables may be unused in some calculators.
e) **Double Data Element**
   A struct with a string and a pointer to a double. Its uses will be elaborated in the Interest Rates section.
f) **Date Data Element**
   A struct with a string and a pointer to a Date. Its uses will be elaborated in the Input Manager section.
g) **User File Path**
   A string containing the file path of the user's profile.

There are a couple of auxillary/helper/secondary functions in DM which do menial tasks, like checking whether a string contains any non-numeric characters.

a) **Is Empty**

`bool IsEmpty(const char *s)`

➢ Checks whether any characters other than the whitespace is present in a char pointer using the isspace() function from the ctype.h library. If it is empty, a true boolean is return.

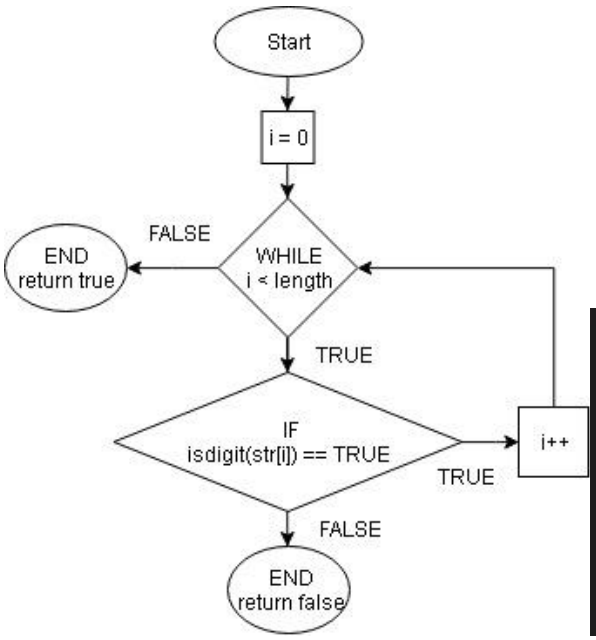



```
Is_Empty
    DOWHILE *s!='\0'
        IF isspace(*s) == FALSE
            return false
        ENDIF
        *s++
    ENDDO
    return true
END
```

b) **Is Integer**

`bool IsInteger(char str[])`

➢ Checks whether a string has any non-numeric character using the isdigit() function from the ctype.h library. It returns a true boolean if all are numeric.

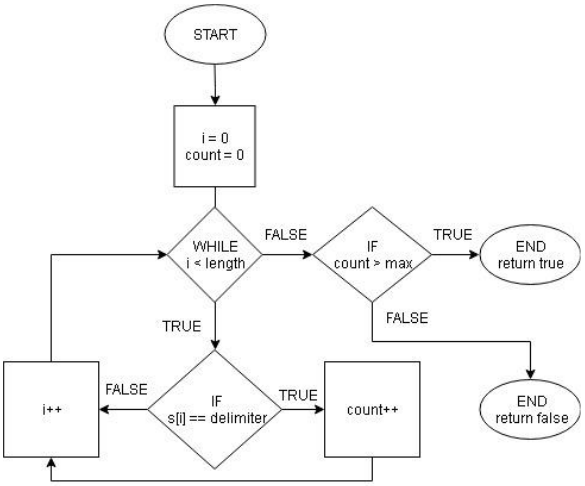



```
Is_Integer
    i = 0
    REPEAT
        IF isdigit(str[i]) == FALSE
            return false
        ENDIF
    UNTIL i >= max
    return true
END
```

c) **Exceed Char Limit**

`bool ExceedCharLimit(char s[], char delimiter, int max)`

➢ Checks whether a particular char, called the delimiter, is present more than a certain number of times in a string. The function returns a true boolean if the frequency exceeds the specified limit/maximum.



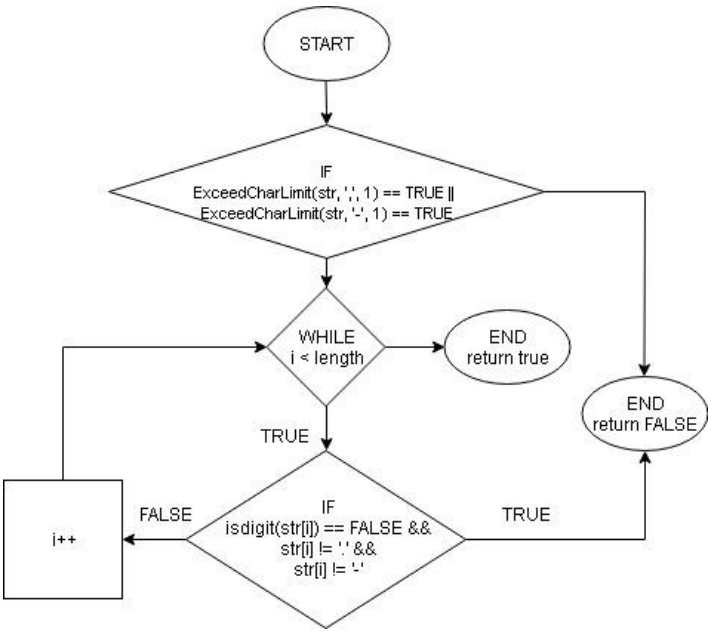

```
Exceed_Char_Limit
    i = 0
    count = 0
    REPEAT
        IF str[i] == delimiter
        count++
        ENDIF
    UNTIL i >= length
    lF count > max
        return TRUE
    ENDIF
    return FALSE
END
```

---

d) **Is Double**

`bool IsDouble(char str[])`

➢ Checks whether a string is an entirely valid double, using the isdigit function from the the ctype.h library and the ExceedCharLimit function. It also ensures that the string does not contain more than one decimal separator or the negative sign. It returns a true boolean if the criteria is fulfilled.



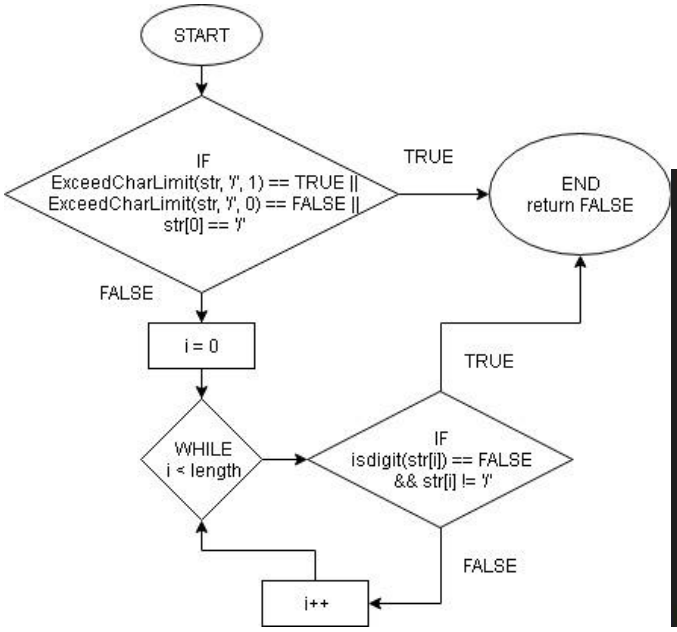

```
Is_Double
    IF ExceedCharLimit(str, ',', 1)
        || ExceedCharLimit(str, '-', 1)
        return FALSE
    ENDIF
    REPEAT
        IF isdigit(str[i]) == FALSE
            && str[i] != '.'
            && str[i] != '-'
            return FALSE
        ENDIF
    UNTIL i >= length

    return TRUE
END
```

---

e) **Is Date**

`bool IsDate(char str[])`

➢ Checks whether a string can be parsed into a Date.

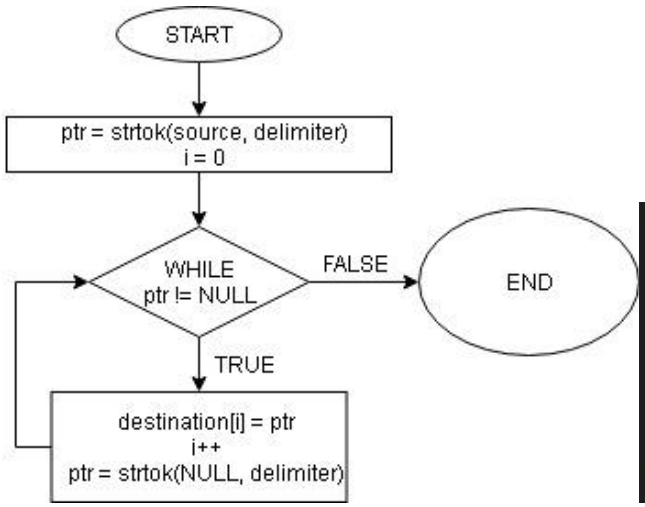



```
Is_Date
    IF ExceedCharLimit(str, '/', 1)
        || !ExceedCharLimit(str, '/', 0)
        || str[0] == '/'
        return FALSE
    ENDIF
    i = 0
    REPEAT
        IF isdigit(str[i]) == FALSE
            && str[i] != '/'
            return FALSE
        ENDIF
    UNTIL i >= length
    ENDIF
    return TRUE
END
```

f) **Split String**

`void SplitString(char* destination[], char source[], char delimiter[])`

➢ Splits a string into an array of strings. The delimiter is the char which indicates the splitting point. The function strtok() from the string.h library is used. (How?)

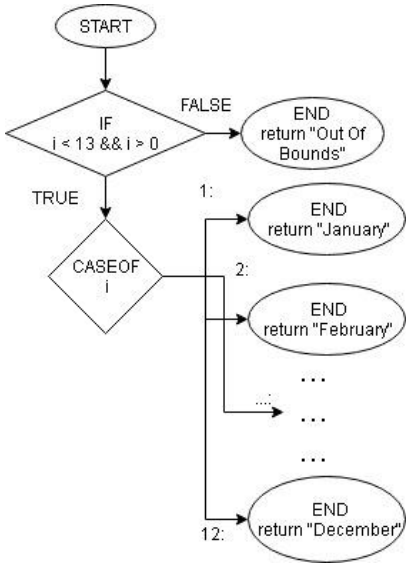



```
Split_String
    ptr = strtok(source, delimiter)
    i = 0
    DOWHILE ptr != NULL
        destination[i] = ptr
        i++
        ptr = strtok(NULL, delimiter)
    ENDDO
END
```

g) **Int To Months**

`char* IntToMonth(int i)`

➢ Returns the name of the month which corresponds to the integer received. It uses a simple, tedious switch-case structure.

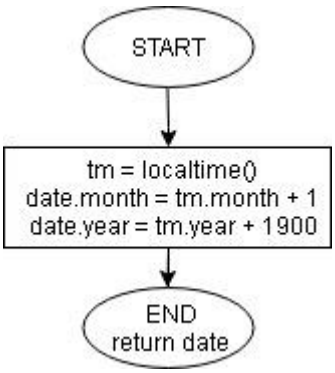



```
Int_To_Month
    IF i < 13 && i > 0
        CASEOF i
            1: return "January"
            2: return "February"
            ...
            12: return "December"
        ENDCASE
    ELSE
        return "Out of bounds!"
END
```

h) **Get Current Date**

`struct Date GetCurrentDate()`

➢ Obtains the current date and converts it into a Date struct. The localtime() function from the time.h library returns a struct containing the current date. The month value of the struct starts from 0 so it is increased by 1. Besides, The year value starts from 1900 so 1900 is added to obtain the current year.

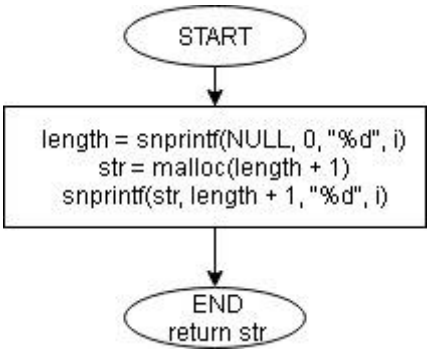



```
Get_Current_Date
    tm = localtime()
    date.month = tm.month + 1
    date.year = tm.year + 1900
    return date
END
```

i) **Int To String**

`char* IntToString(int i)`

➢ Converts an integer to a string. The snscanf() function from the stdio.h library is used to obtain the length of the integer and also convert it to a string. malloc() is used to allocate a fixed chunk of memory for the string. The memory size depends on the integer length but an extra byte is added for the string terminator.




```
Get_Current_Date
    length = snprintf(NULL, 0, "%d", i)
    str = malloc(length + 1)
    snprintf(str, length + 1, "%d", i)
    return str
END
```

j) **Open FAQ**

`void OpenFAQ(enum FAQTypes type)`

➢ Opens an Internet link which corresponds to the enumerator received. To open the link, system("cmd /c start link") is used.

k) **Draw Graph**

`void DrawGraph(char title[], void (*display)(), double(*func)(double), double xMax, char* xTitle, double yOffset, double yMax, char* yTitle)`

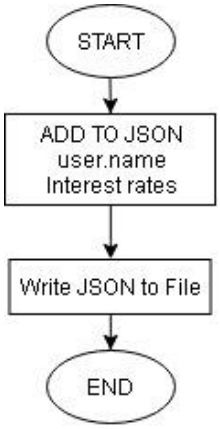
➢ Displays a basic graph with two perpendicular axes. Each axis has five divisions, meaning that the entire axis which spans a maximum and a minimum is divided into five equal intervals. The x axis is treated as the domain while the y axis is the range. A function pointer references to a function which computes the y values from the x values. Using such a pointer enables different sections of the program to utilise this function with different computation methods. In other words, the function is generalised such that it can accommodate various forms of calculations. This design has saved us effort and time by a great margin, as we need not create redundant graphing functions tailored to each section. As the algorithm involved is too complex, we have decided not to construct visual diagrams for it.

The primary functions of DM are catered to manipulating the user's data.

a) **Update User Data**

`void data_UpdateUserData(bool reset)`

➢ Updates the user's file with the relevant values of the user's variables. The cJSON library is used to encode the data in JSON. Default values are used if the boolean is true. The flowchart is an oversimplification. Please refer to the original code for the technical details.

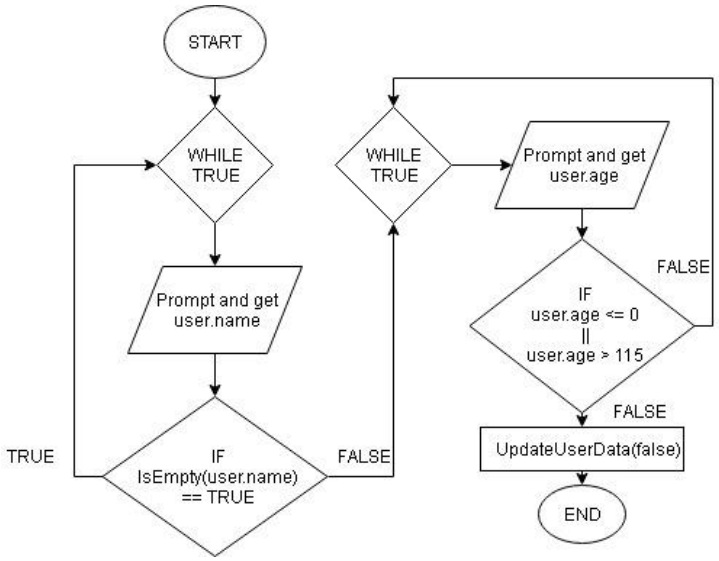



```
Update_User_Data
    ADDTOJSON(user.name, Interest rates)
    WriteFile(JSON)
END
```

b) **First Time**

`void data_FirstTime()`

➢ Asks for the user's name and age. Non-numeric characters are not checked for the age as it is not used in any other parts of the program. In fact, these two parameters are practically useless and are for fun's sake.

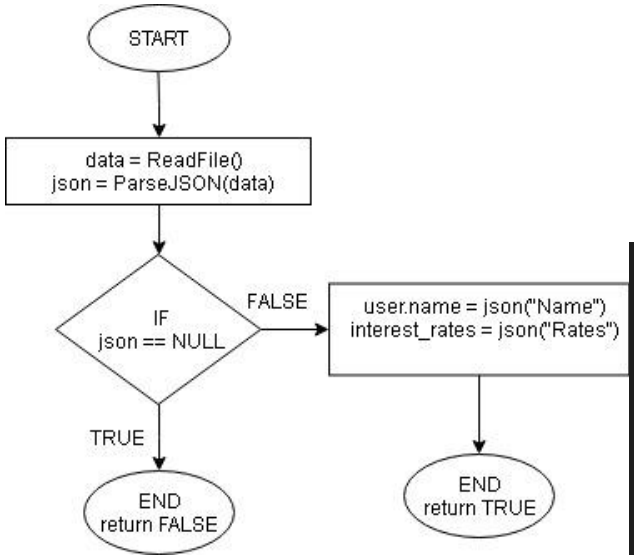



```
First_Time
    DOWHILE TRUE
        Prompt and Get user.name
        IF IsEmpty(user.name) == FALSE
            break
        ENDIF
    ENDDO
    DOWHILE TRUE
        Prompt and Get user.age
        IF user.age > 0 && user.age <= 115
            break
        ENDIF
    ENDDO
    UpdateUserData(false)
END
```

c) **Load User Data**

`bool data_LoadUserData()`

➢ Interprets the user's profile and restore the user's values in the program. It returns a false boolean if an error is encountered during the parsing. The flowchart is an oversimplification. Please refer to the original code for the technical details.




```
Load_User_Data
    data = ReadFile()
    json = ParseJSON(data)
    IF json == NULL
        return FALSE
    user.name = json("Name")
    interest_rates = json("Rates")
    return TRUE
END
```

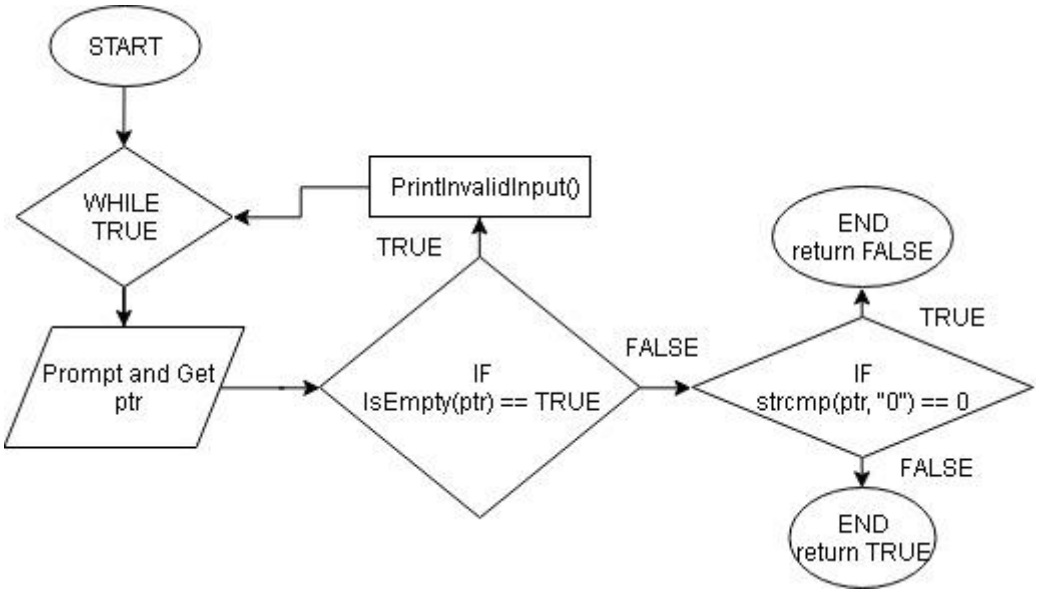
# 1.4: Input Manager

To minimise redundant code for user inputs, we constructed a template of functions under the heading, Input Manager, abbreviated as IM. These functions cater to specific types of user inputs layered with simple yet effective error-checking mechanisms. Some of them return a false boolean if the input string only has a single '0' char. This is because, in this program, the char indicates the user's intention to exit. Via this sentinel value, the caller of the function will be able to detect the intention and enforce whatever is necessary. Otherwise, the functions always return true. sscanf() and strcmp() from the stdio.h and the string.h library respectively are frequently used.

a) **Input String**

```
bool input_String(char* title, char query[], char* ptr)
```

➢ Since strings are almost universal, there are no errors to check for.

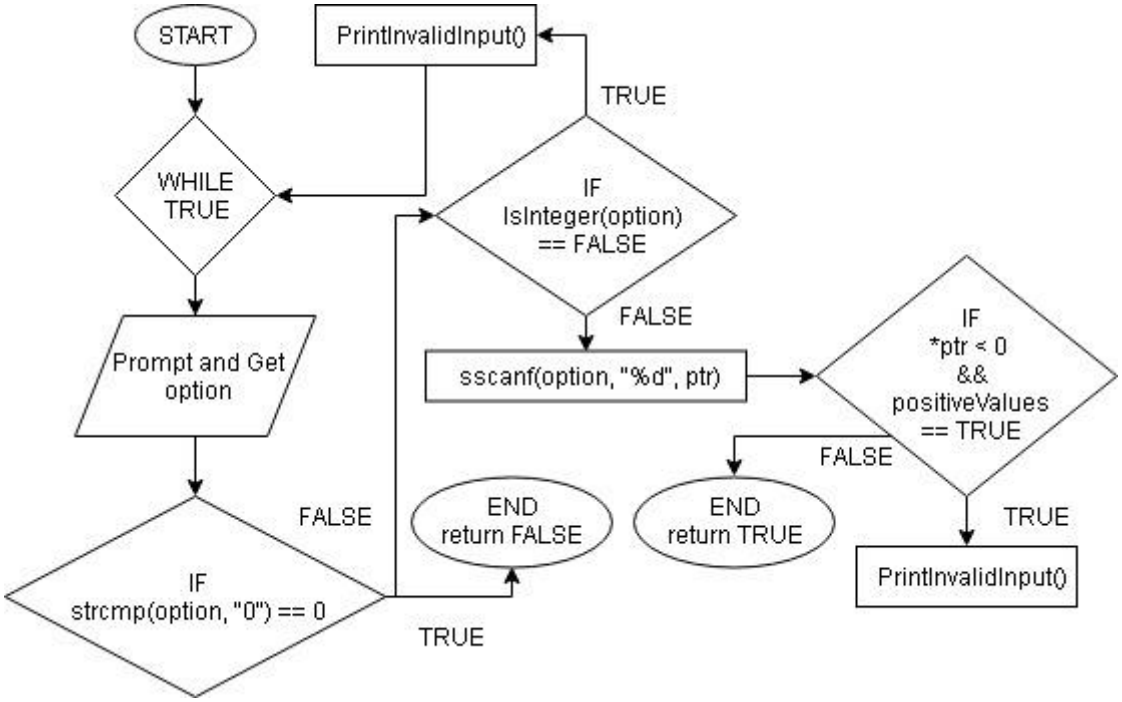



```
INPUT_STRING
    DOWHILE TRUE
        Prompt and Get ptr
        IF IsEmpty(ptr) == TRUE
            PrintInvalidInput()
        ELSE
            IF strcmp(ptr, "0") == 0
                return FALSE
            ELSE
                return TRUE
            ENDIF
        ENDIF
    ENDDO
END
```

b) **Input Int**

```
bool input_Int(char* title, char query[], int* ptr, bool positiveValues)
```

➢ If the "positiveValues" boolean is true, the function will safeguard against negative integers.

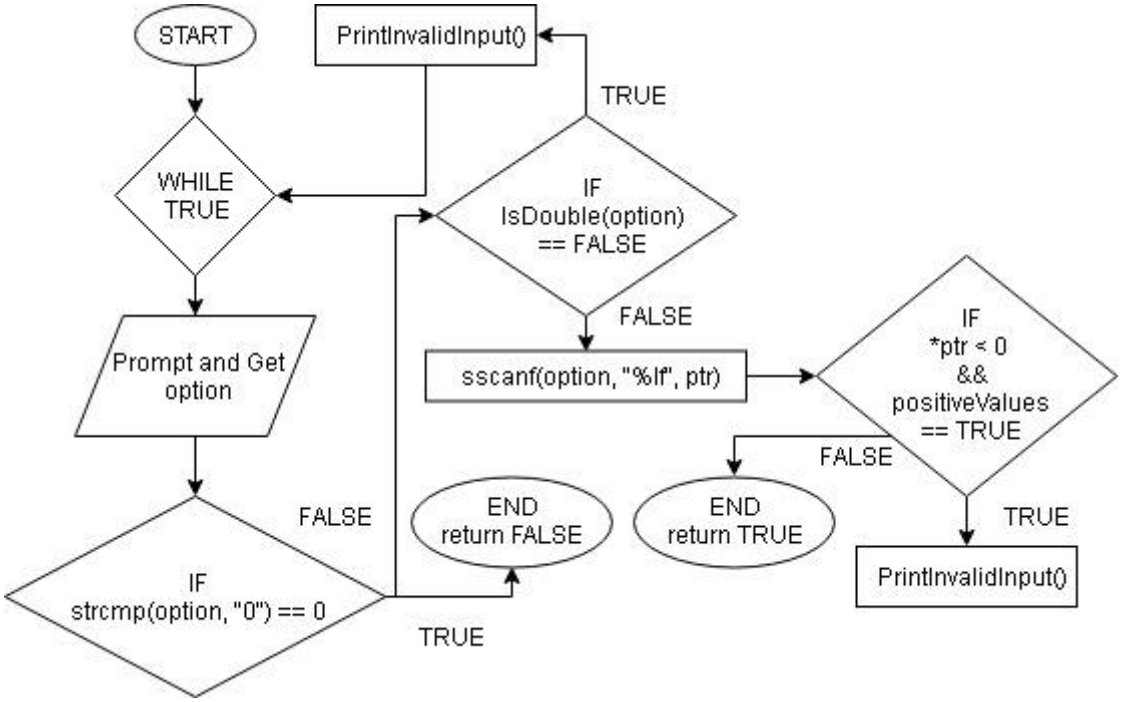



```
INPUT_INT
    DOWHILE TRUE
        Prompt and Get option
        IF strcmp(option, "0") == 0
            return FALSE
        ENDIF
        IF IsInteger(option) == FALSE
            PrintInvalidInput()
        ELSE
            sscanf(option, "%d", ptr)
            IF *ptr < 0 && positiveValues == TRUE
                PrintInvalidInput()
            ELSE
                return TRUE
        ENDIF
    ENDDO
END
```

c) **Input Double**

`bool input_Double(char* title, char query[], double* ptr, bool positiveValues)`

➢ If the "positiveValues" boolean is true, the function will safeguard against negative doubles.

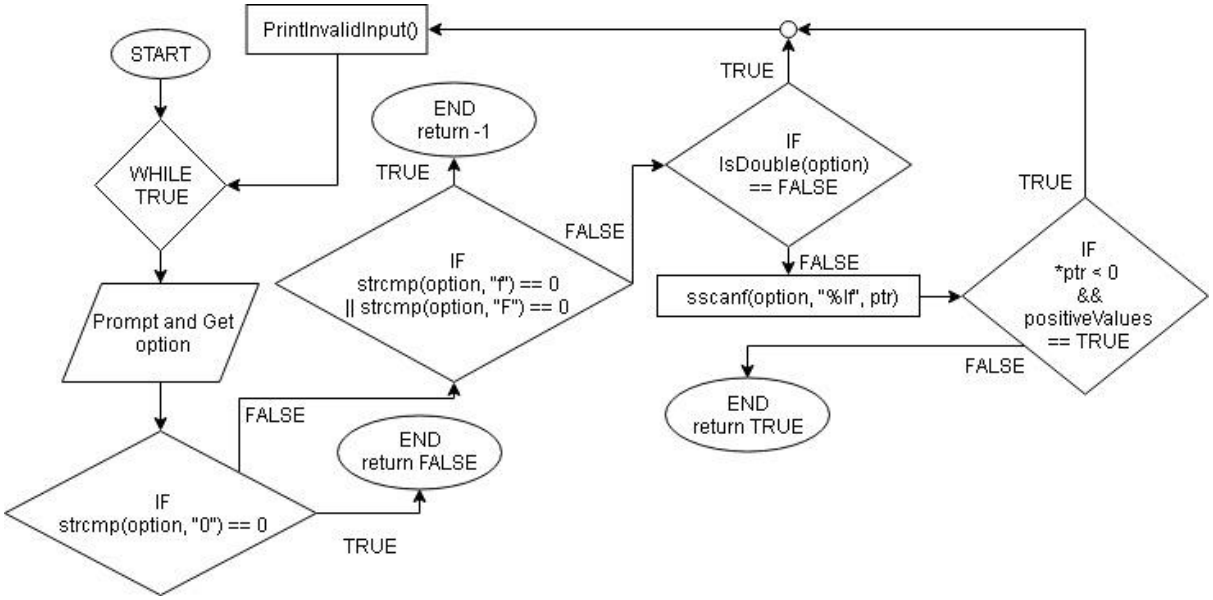



```
Input_Double
    DOWHILE TRUE
        Prompt and Get option
        IF strcmp(option, "0") == 0
            return FALSE
        ENDIF
        IF IsDouble(option) == FALSE
            PrintInvalidInput()
        ELSE
            sscanf(option, "%lf", ptr)
            IF *ptr < 0 && positiveValues == TRUE
                PrintInvalidInput()
            ELSE
                return TRUE
            ENDIF
        ENDIF
    ENDDO
END
```

d) **Input Double List**

`int input_Double_List(char* title, char query[], double* ptr, bool positiveValues)`

➢ Although similar to Input Double, this function also checks for the char 'F' / 'f'. These two chars indicate that the user wants to stop adding to a linked list. -1 is returned for 'F' whereas a false boolean is returned for '0' as usual. If the "positiveValues" boolean is true, the function will safeguard against negative doubles.

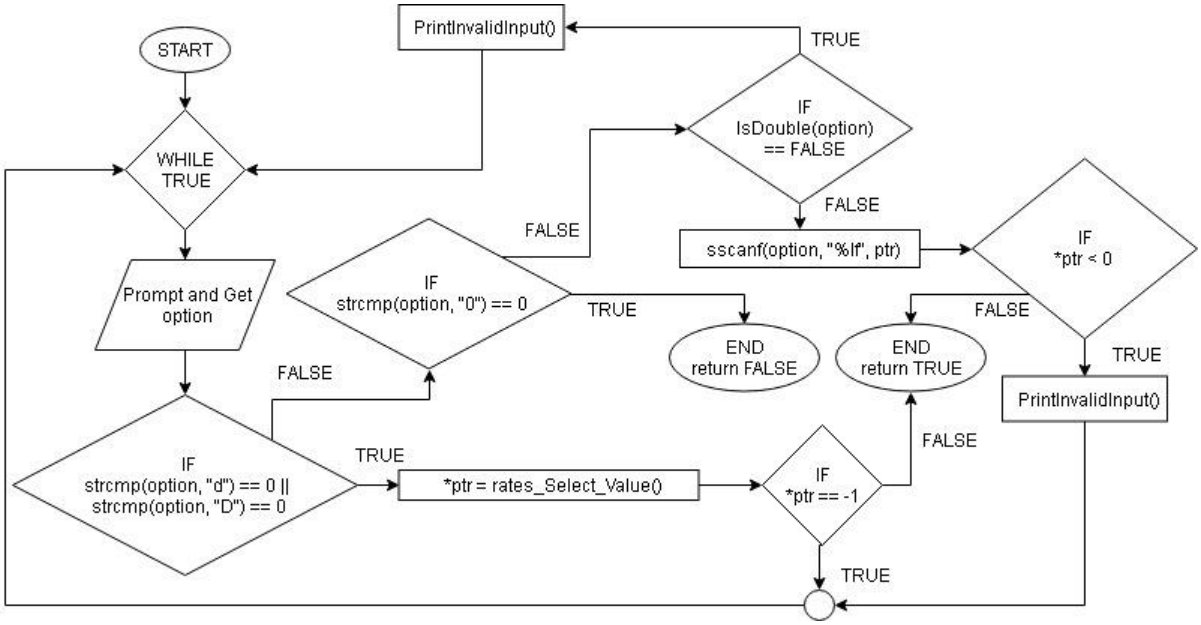



```
Input_Double_List
    DOWHILE TRUE
        Prompt and Get option
        IF strcmp(option, "0") == 0
            return FALSE
        ENDIF
        IF strcmp(option, "f") == 0
        || strcmp(option, "F") == 0
            return -1
        ENDIF
        IF IsDouble(option) == FALSE
            PrintInvalidInput()
        ELSE
            sscanf(option, "%lf", ptr)
            IF *ptr < 0 && positiveValues == TRUE
                PrintInvalidInput()
            ELSE
                return TRUE
            ENDIF
        ENDIF
    ENDDO
END
```

e) **Input Interest**

`bool input_Interest(char* title, double* ptr)`

➢ Retrieves an interest rate from either a user-specified value or the user's profile/database. The char 'D'/'d' reflects the user's intention to access the database. Rates_Select_Value() is the function which facilitates the process and returns -1 if the user does not select a value from the database. Otherwise, any numerical input will be regarded as the desired interest rate.




```
Input_Interest
    DOWHILE TRUE
        Prompt and Get option
        IF strcmp(option, "d") == 0
        || strcmp(option, "D") == 0
            *ptr = rates_Select_Value()
            IF *ptr == -1
                continue
            ELSE
                return TRUE
            ENDIF
        ENDIF
        IF strcmp(option, "0") == 0
            return FALSE
        ENDIF
        IF IsDouble(option) == FALSE
            PrintInvalidInput()
        ELSE
            sscanf(option, "%lf", ptr)
            IF *ptr < 0
                PrintInvalidInput()
            ELSE
                return TRUE
            ENDIF
        ENDIF
    ENDDO
END
```

f) **Input Date Duration**

```
int input_Date_Duration(int month1, int year1, int month2, int year2)
```

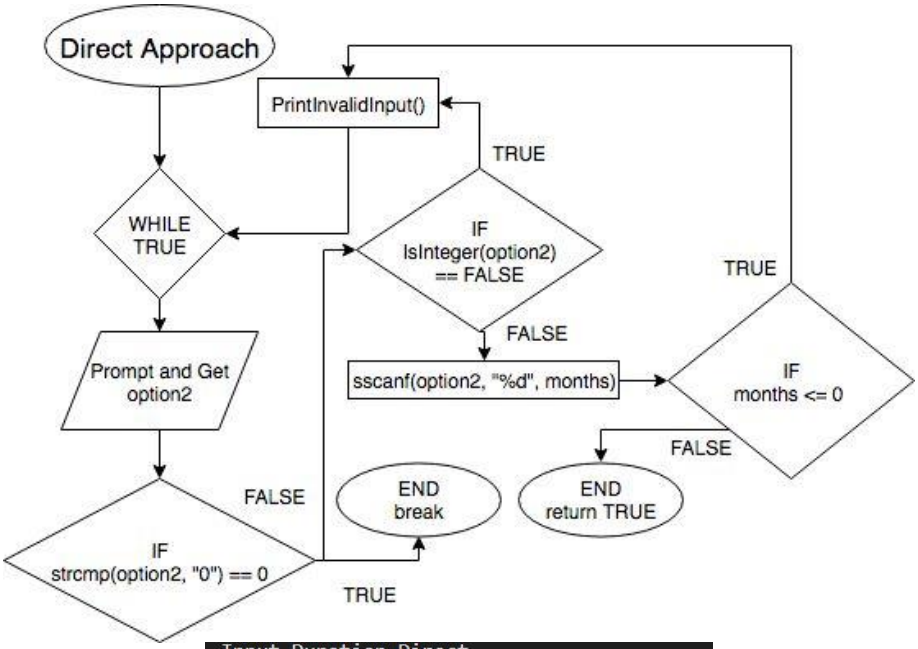
➢ Computes the duration(in months) between two dates.

g) **Input Duration**

```
bool input_Duration(char* title, struct Duration *duration)
```

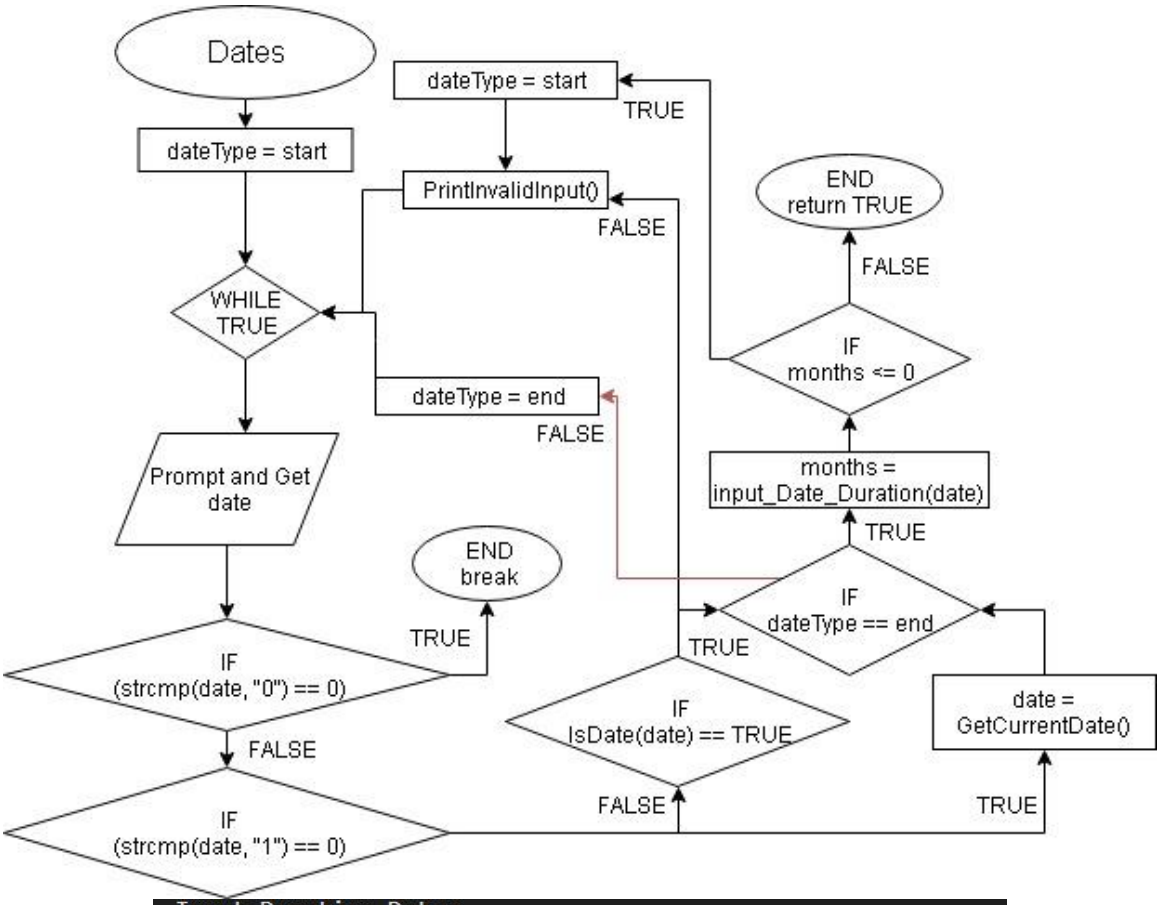
➢ Obtains a user-specified duration, either in direct numerical form or in a formatted manner. The direct approach can only be used for the month variable in the Duration struct. To obtain a value for the year variable instead, Input Integer is used. The formatted approach evaluates the number of months from two dates given by the user. The dates are rigidly defined in the following manner: month/year. The user can also opt for the current date. The second date must be chronologically "later" than the first. To make it easier to visualise, we split the function into three parts. (The diagrams only contain vague descriptions of the actual code.)

i. Direct Approach



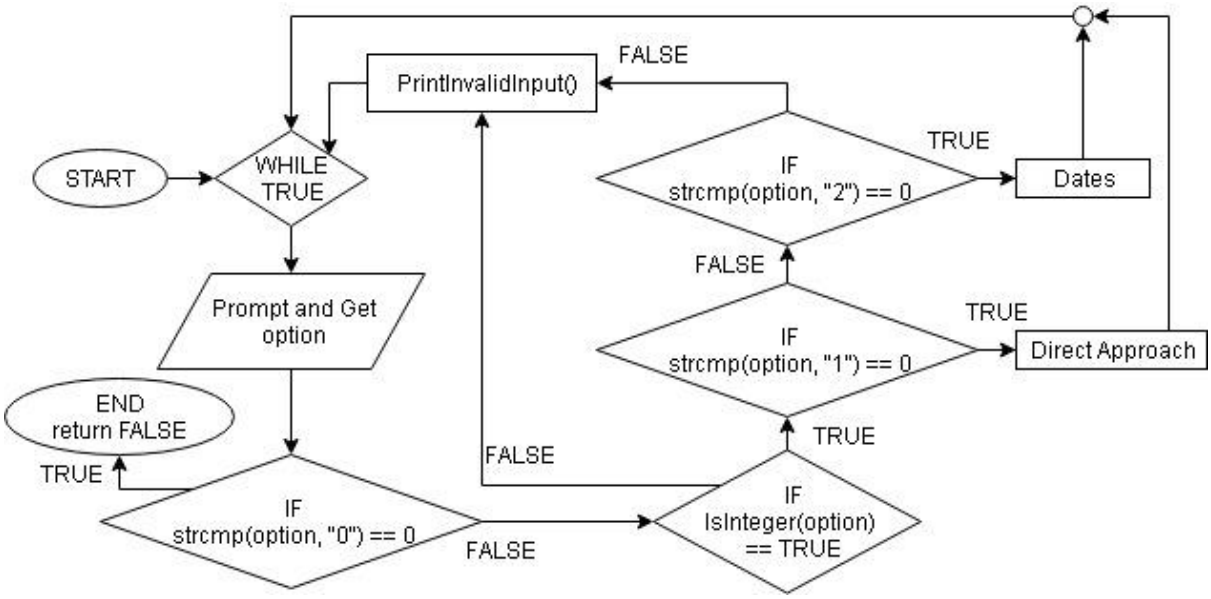

```
Input_Duration_Direct
    DOWHILE TRUE
        Prompt and Get option2
        IF strcmp(option2, "0") == 0
            break
        ENDIF
        IF IsInteger(option2) == FALSE
            PrintInvalidInput()
        ELSE
            sscanf(option2, "%d", months)
            IF months <= 0
                PrintInvalidInput()
            ELSE
                return TRUE
            ENDIF
        ENDIF
    ENDDO
END
```

ii. Formatted Approach / Dates




```
Input_Duration_Dates
    dateType = start
    DOWHILE TRUE
        Prompt and Get date
        IF strcmp(date, "0") == 0
            break
        ENDIF
        IF strcmp(date, "1") == 0
            date = GetCurrentDate()
        IF IsDate(date) == TRUE || strcmp(date, "1") == 0
            IF dateType == end
                months = input_Date_Duration(date)
                IF months <= 0
                    dataType = start
                    PrintInvalidInput()
                ELSE
                    return TRUE
                ENDIF
            ELSE
                dataType = end
            ENDIF
        ELSE
            PrintInvalidInput()
        ENDIF
    ENDDO
END
```

iii. Main



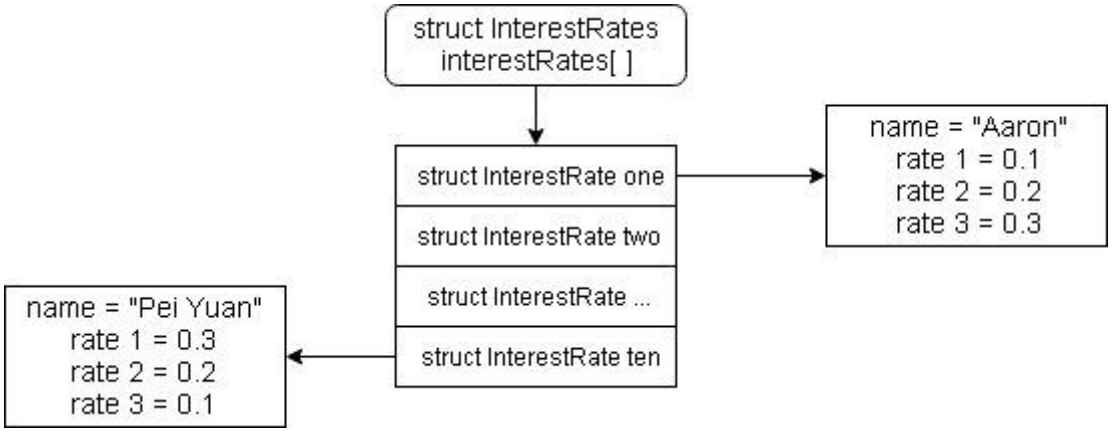

```
Input_Duration_Main
    DOWHILE TRUE
        Prompt and Get option
        IF strcmp(option, "0") == 0
            return TRUE
        ENDIF
        IF IsInteger(option) == TRUE
            IF strcmp(option, "1") == 0
                Direct_Approach
            ELSEIF strcmp(option, "1") == 0
                Dates
            ELSE
                PrintInvalidInput()
            ENDIF
        ELSE
            PrintInvalidInput()
        ENDIF
    ENDDO
END
```

## 1.5: Interest Rates

If the reader has gone through the previous subsections, he/she should be very much aware by now of how imperative this component is to the program. We will abbreviate it as IR. This section discusses how the database maintains itself and also how it complements other components seamlessly.

```
#define DEFAULT_INTEREST_RATES_MAX 10
struct InterestRate{
    char name[DEFAULT_CHAR_MAX];
    double rate1;
    double rate2;
    double rate3;
} interestRates[DEFAULT_INTEREST_RATES_MAX];
```

The user's interest rates are stored in an array of structs. Each struct, called an **interest group**, comprises of a **name** with **three double variables** referred to as interest rates. It is a consensus among our team that the **maximum** structs should be **10**, and that each struct should only hold **three interest rates**. Consequently, the user can store up to **30** rates and nothing more. Although the interest groups are already allocated in the memory at program initialisation, they are regarded as non-existent if their name variable is empty (only has the null character, "\0"). The diagram below describes the data structure:




In DM, a struct array variable named rateElements is declared there. This array comprises of DoubleDataElement structs, each storing a name and a pointer to a double. These structs actually correspond to the three interest rates in an interest group. Hence, the array only has three element structs. Their purpose is to temporarily store the interest rate values of a particular interest group. It may not be obvious, but the overall code is reduced when they are utilised in a certain generalised manner which will be explained in later sections.

The following are the secondary functions of the component:

a) **Reset Rates**

`void rates_ResetRate(int index)`

➢ Removes all user-defined interest rates.

b) **Display Length**

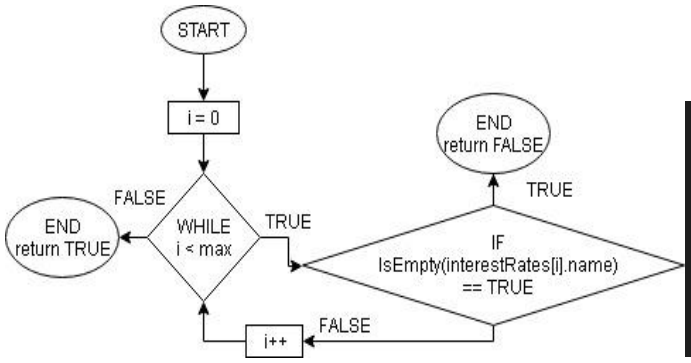
`int rates_DisplayLength()`

➢ Returns an integer which depends on the number of interest groups.

c) **Is Full**

`bool rates_IsFull()`

➢ Returns a true boolean if the user has created the maximum number of interest groups.

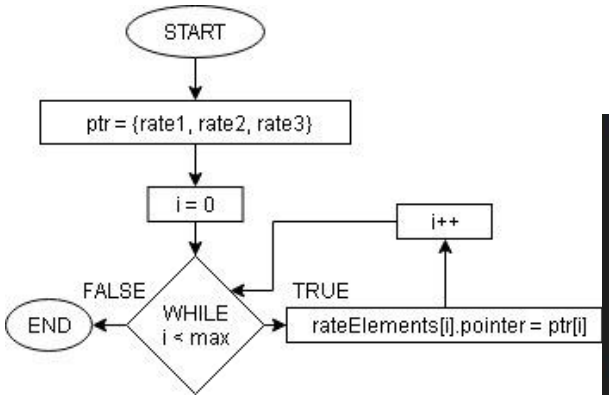



```
Is_Full
    i = 0
    REPEAT
        IF IsEmpty(interestRates[i].name) == TRUE
            return FALSE
        ENDIF
        i++
    UNTIL i >= max
    return TRUE
END
```

d) **Update Data Elements**

`void rates_UpdateDataElements(int index)`

➢ Stores the values of an interest group in the rateElements variable.

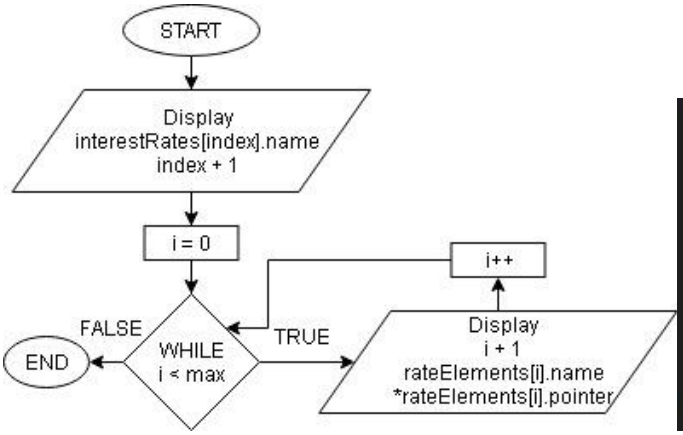



```
Update_Data_Elements
    ptr = {rate1, rate2, rate3}
    i = 0
    REPEAT
        rateElements[i].pointer = ptr[i]
        i++
    UNTIL i >= max
END
```

e) **Display Element Index**

`void rates_DisplayElement_Index(int index)`

➢ Displays the name of an interest group and its interest rates with their index.

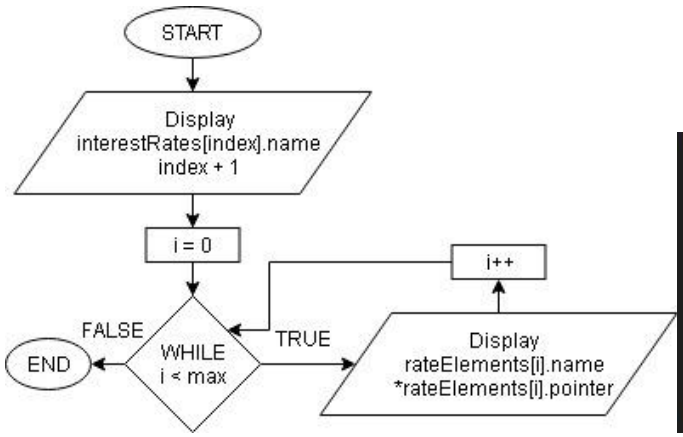



```
Display_Element_Index
    Display interestRates[index].name
    Display index + 1
    i = 0
    REPEAT
        Display i + 1
        Display rateElements[i].name
        Display *rateElements[i].pointer
        i++
    UNTIL i >= max
END
```

f) **Display Element**

`void rates_DisplayElement(int index)`

➢ Displays the name of an interest group and its interest rates in horizontally.

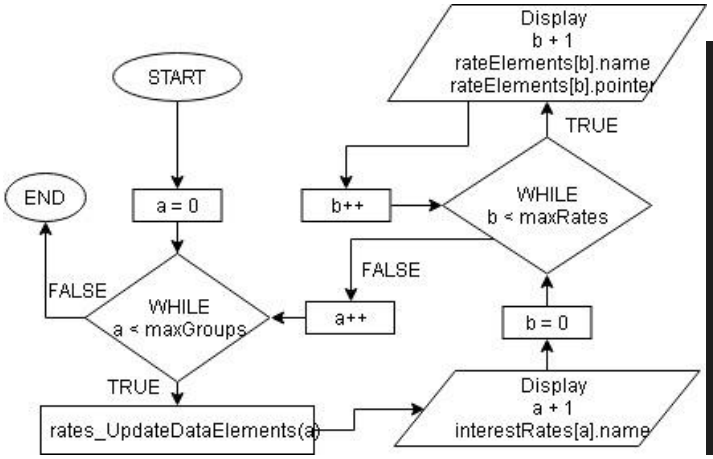



```
Display_Element
    Display interestRates[index].name
    Display index + 1
    i = 0
    REPEAT
        Display rateElements[i].name
        Display *rateElements[i].pointer
        i++
    UNTIL i >= max
END
```

g) **Display**

`void rates_Display()`

➢ Displays the user's interest groups and their interest rates in a table. This function modifies the rateElements as it iterates through the interest groups.

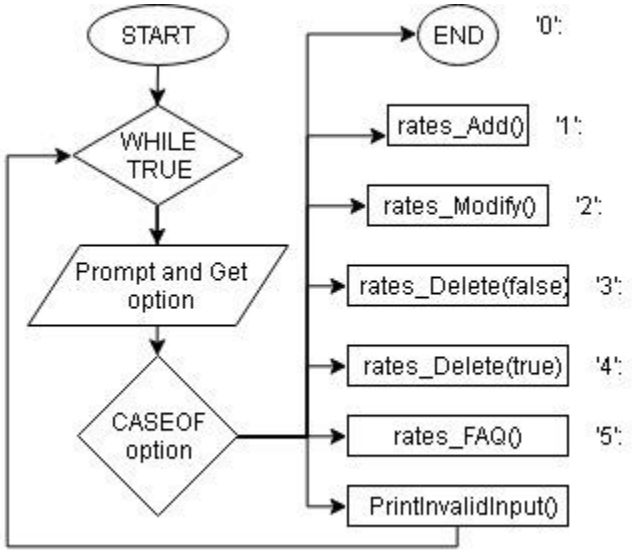



```
Display
    a = 0
    REPEAT
        rates_UpdateDataElements(a)
        Display a + 1
        Display interestRates[index].name
        b = 0
        REPEAT
            b + 1
            Display rateElements[i].name
            Display *rateElements[i].pointer
            b++
        UNTIL b >= maxRates
        a++
    UNTIL a >= maxGroups
END
```

h) **Open Interest Rates**

`void OpenInterestRates()`

➢ Opens the main menu for this section.



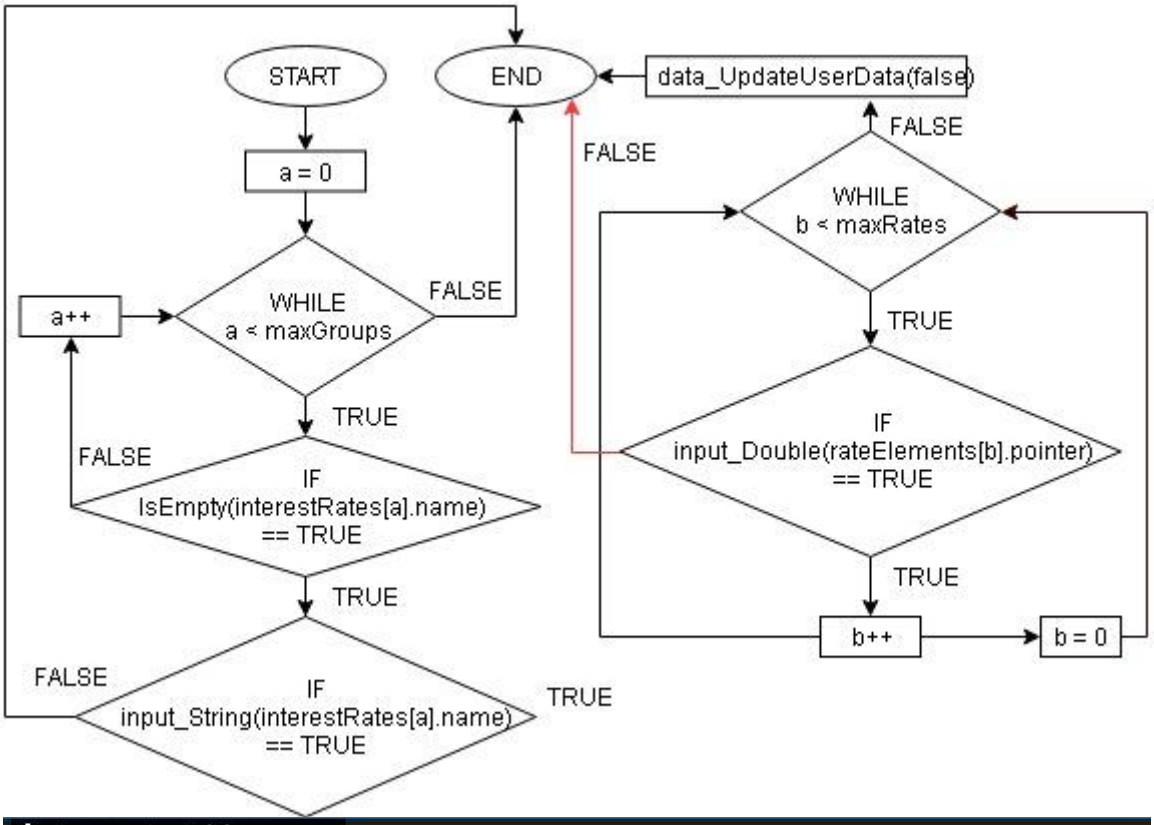

```
Open_Interest_Rates
    DOWHILE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': rates_Add()
            '2': rates_Modify()
            '3': rates_Delete(false)
            '4': rates_Delete(true)
            '5': rates_FAQ()
           others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

The primary functions are as follows:

a) **Add**

`void rates_Add()`

➢ Adds an interest group to the database.

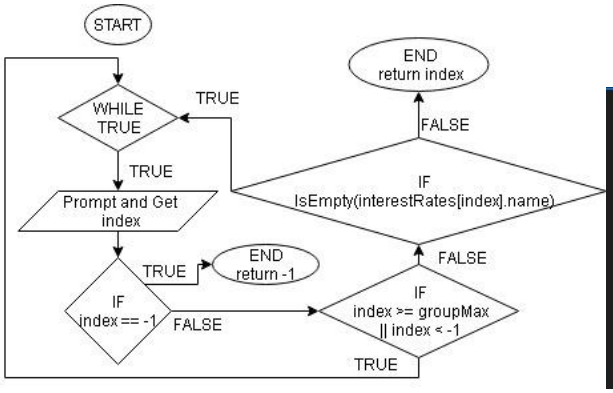



```
interest_Add
    a = 0
    REPEAT
        IF IsEmpty(interestRates[a].name) == TRUE
            IF input_String(interestRates[a].name) == TRUE
                b = 0
                REPEAT
                    IF input_Double(rateElements[b].pointer) == TRUE
                        b++
                    ELSE
                        return
                UNTIL b >= maxRates
                data_UpdateUserData(false)
                return
            ELSE
                return
            ENDIF
        ENDIF
    UNTIL a >= maxGroups
END
```

b) **Select Index**

`int rates_Select_Index(char title[])`

➢ Returns the index of the selected interest group.

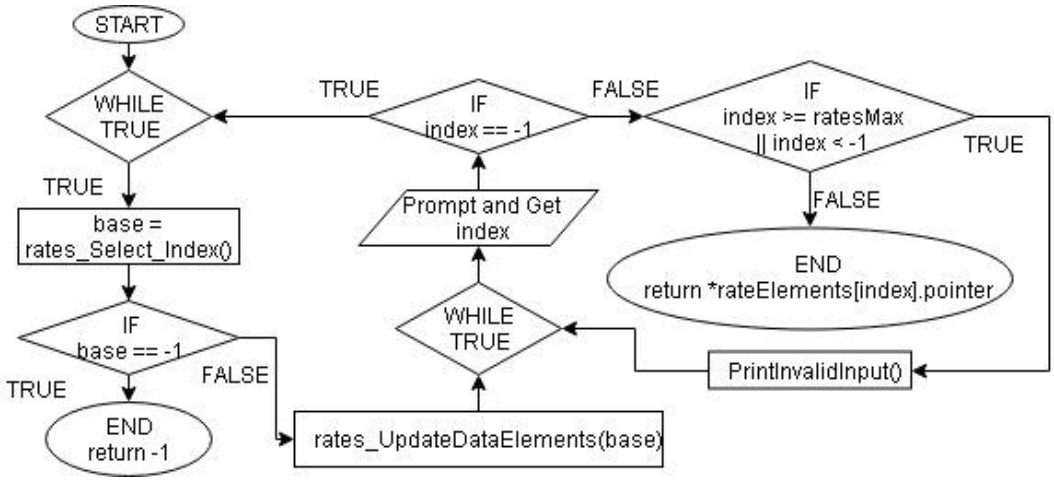



```
Select_Index
    DOWHILE
        Prompt and Get index
        IF index == -1
            return -1
        ENDIF
        IF index >= groupMax || index < -1
            IF IsEmpty(interestRates[index].name) == FALSE
                return index
            ENDIF
        ENDIF
    ENDDO
END
```

c) **Select Value**

`double rates_Select_Value(char title[])`

➢ Returns the selected Interest rate of a particular interest group.

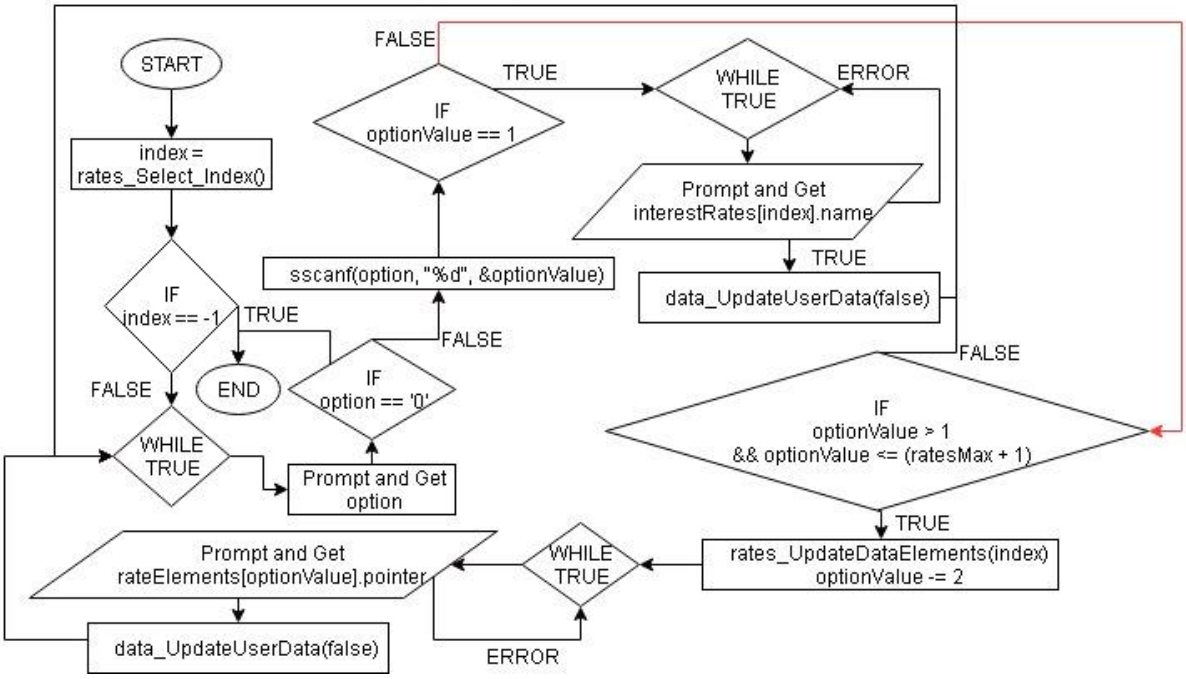



```
Select_Value
    DOWHILE
        base = rates_Select_Index()
        IF base == -1
            return -1
        ENDIF
        rates_UpdateDataElements(base)
        DOWHILE
            Prompt and Get index
            IF index == -1
                break
            ENDIF
            IF index >= ratesMax || index < -1
                PrintInvalidInput()
            ELSE
                return *rateElements[index].pointer
            ENDIF
        ENDDO
    ENDDO
END
```

d) **Modify**

`void rates_Modify()`

➢ Modifies a selected interest group. Some error-checking mechanisms are not shown in the diagrams.

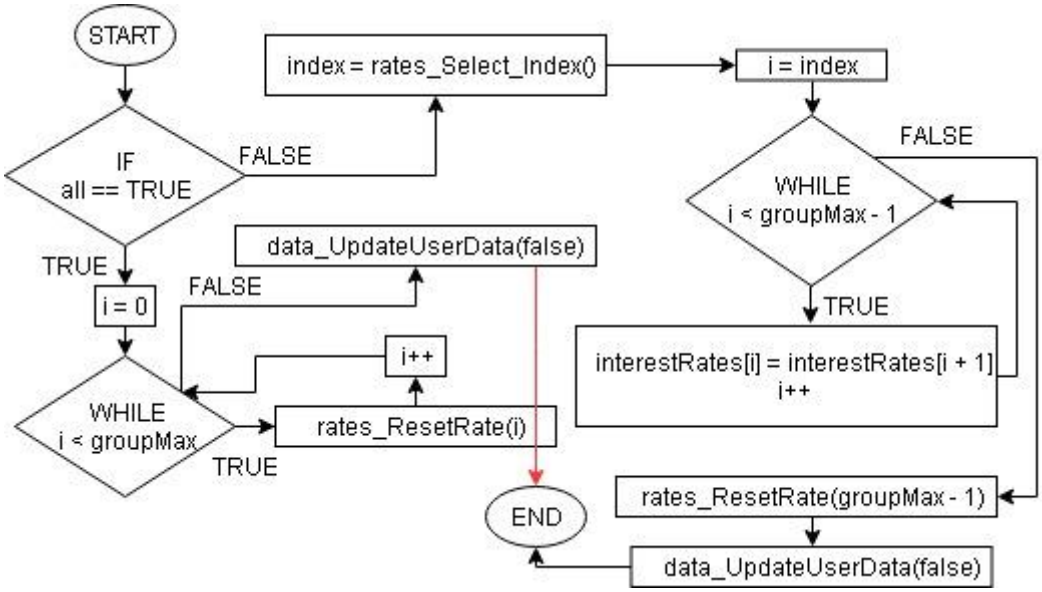



```
interest_Modify
    index = rates_Select_Index()
    IF index == -1
        return
    ENDIF
    DOWHILE
        Prompt and Get option
        IF option == '0'
            return
        ENDIF
        sscanf(option, "%d", &optionValue)
        IF optionValue == 1
            Prompt and Get interestRates[index].name
            data_UpdateUserData(false)
        ELSEIF optionValue > 1 && optionValue <= (ratesMax + 1)
            rates_UpdateDataElements(index)
            optionValue -= 2
            Prompt and Get rateElements[optionValue].pointer
            data_UpdateUserData(false)
        ENDIF
    ENDDO
END
```

e) **Delete**

`void rates_Delete(bool all)`

➢ If the boolean is true, the function resets the interest groups to default. Otherwise, it resets/removes only a particular group. The groups located after the removed group will be shifted towards the start.




```
interest_Delete
    IF all == TRUE
        i = 0
        REPEAT
            rates_ResetRate(i)
            i++
        UNTIL i >= groupMax
        data_UpdateUserData(false)
    ELSE
        index = rates_Select_Index()
        i = index
        REPEAT
            interestRates[i] = interestRates[i + 1]
            i++
        UNTIL i >= groupMax
        rates_ResetRate(groupMax - 1)
        data_UpdateUserData(false)
    ENDIF
END
```

f) **FAQ**

`void rates_FAQ()`

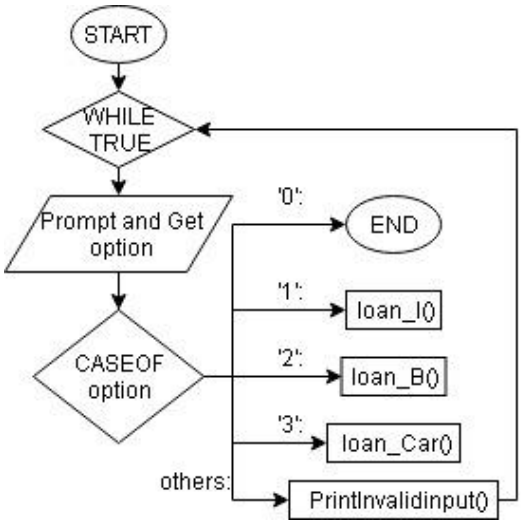
➢ Opens the FAQ menu.

# Part 2: Loans

Loans are financial instruments which comprise of the following essential aspects:

a) Principal
b) Interest rate
c) Duration
d) Net total

Some loans may also include monthly payments and various other factors. For this program, we did not venture far and designed the loans mostly around these four aspects. The OpenLoans() function opens the main menu for this section. The following are diagrams for the function:




```
Open_Loans
    DOWHILE TRUE
        Prompt and Get option
        CASEOF
            '0': return
            '1': loan_I()
            '2': loan_B()
            '3': loan_Car()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

A struct, Loan, contains the variables utilised by the functions in this section.

```
struct Loan {
    double principal;
    double interest;//Annual interest rate
    double monthlyPayment;
    struct Duration duration;
    double total;//The net, which combines both the interest and the principal
} loan; //A variable to store temporary values.
```

# 2.1: Interest Only

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void loan_I_Display_Details()`

➢ Display details regarding the loan.

b) **Forecast**

`void loan_I_Forecast(char title[])`

➢ Displays a list of details for every single month of the loan.

c) **Compute Graph**

`double loan_I_Compute_Graph(double months)`

➢ Used as a function pointer by the graphing function in DM.

d) **FAQ**

`void loan_I_FAQ()`

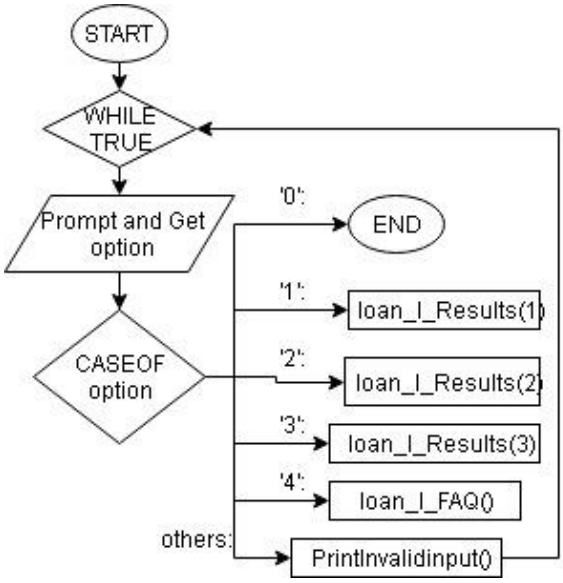
➢ Opens the FAQ menu.

---

e) **Loan I**

`void loan_I()`

➢ Displays the menu for this loan.



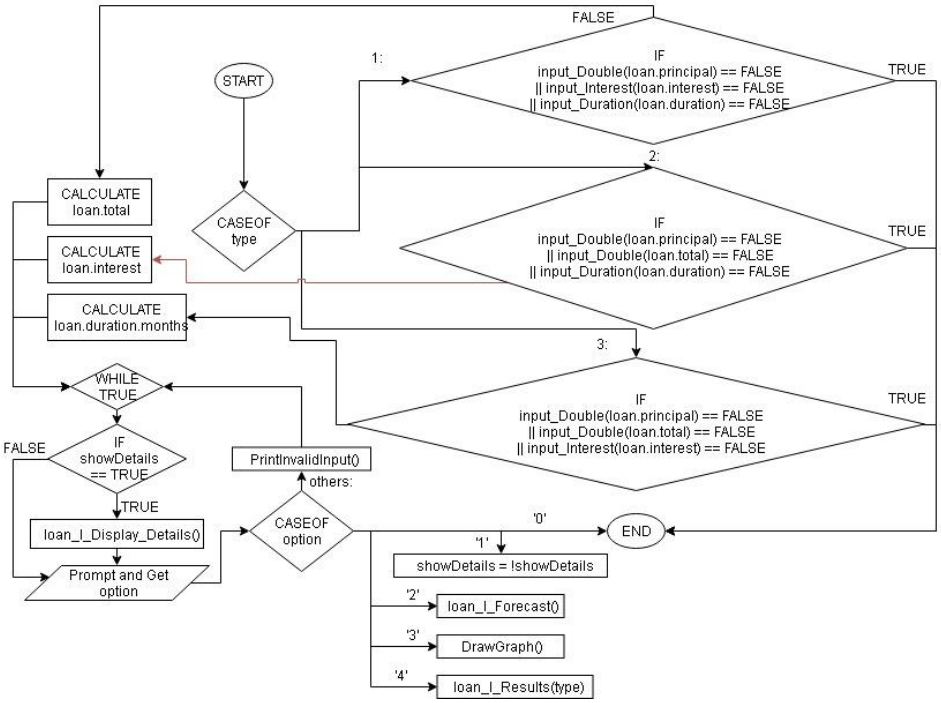

```
loan_I_Display_Details
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': loan_I_Results(1)
            '2': loan_I_Results(2)
            '3': loan_I_Results(3)
            '4': loan_I_FAQ
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

---

f) **Results**

`void loan_I_Results(const int type)`

➢ Execute a range of things depending on the user's option from loan_I().




```
loan_I_Results
    CASEOF type
        1: IF   input_Double(loan.principal) == FALSE
                || input_Interest(loan.interest) == FALSE
                || input_Duration(loan.duration) == FALSE
             return
           ELSE
             CALCULATE loan.total
           ENDIF
        2: IF   input_Double(loan.principal) == FALSE
                || input_Double(loan.total) == FALSE
                || input_Duration(loan.duration) == FALSE
             return
           ELSE
             CALCULATE loan.interest
           ENDIF
        3: IF   input_Double(loan.principal) == FALSE
                || input_Double(loan.total) == FALSE
                || input_Interest(loan.interest) == FALSE

             return
           ELSE
             CALCULATE loan.duration.months
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
          loan_I_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': loan_I_Forecast()
            '3': DrawGraph()
            '4': loan_I_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# 2.2: Interest and Monthly Payments

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void loan_B_Display_Details()`

➢ Display details regarding the loan.

b) **Forecast**

`void loan_B_Forecast(char title[])`

➢ Displays a list of details for every single month of the loan.

c) **Compute Graph**

`double loan_B_Compute_Graph(double months)`

➢ Used as a function pointer by the graphing function in DM.

d) **Compute Monthly Payment**

`double loan_B_Compute_MonthlyPayment(double principal, double annualInterest, double months)`

➢ Returns the monthly payment value as a double.

e) **Compute Interest Rate**

`double loan_B_Compute_InterestRate(double principal, double months, double monthlyPayment, double start, int iterations)`

➢ Computes the interest rate using Newton's approximation.

f) **Compute Duration**

`double loan_B_Compute_Duration(double principal, double monthlyPayment, double annualInterest)`

➢ Returns the duration value as a rounded-up integer.

g) **FAQ**

`void loan_B_FAQ()`

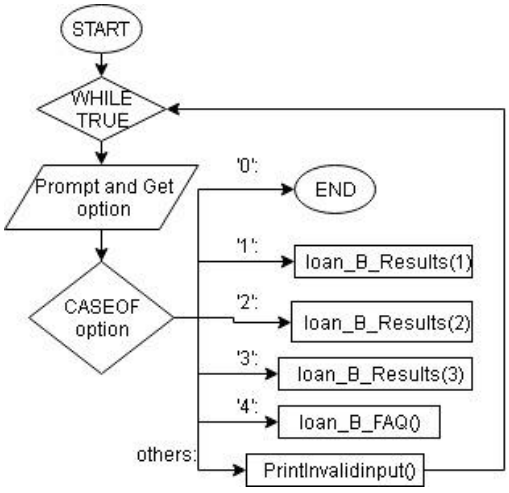
➢ Opens the FAQ menu.

---

h) **Loan B**

`void loan_B()`

➢ Displays the menu for this loan.

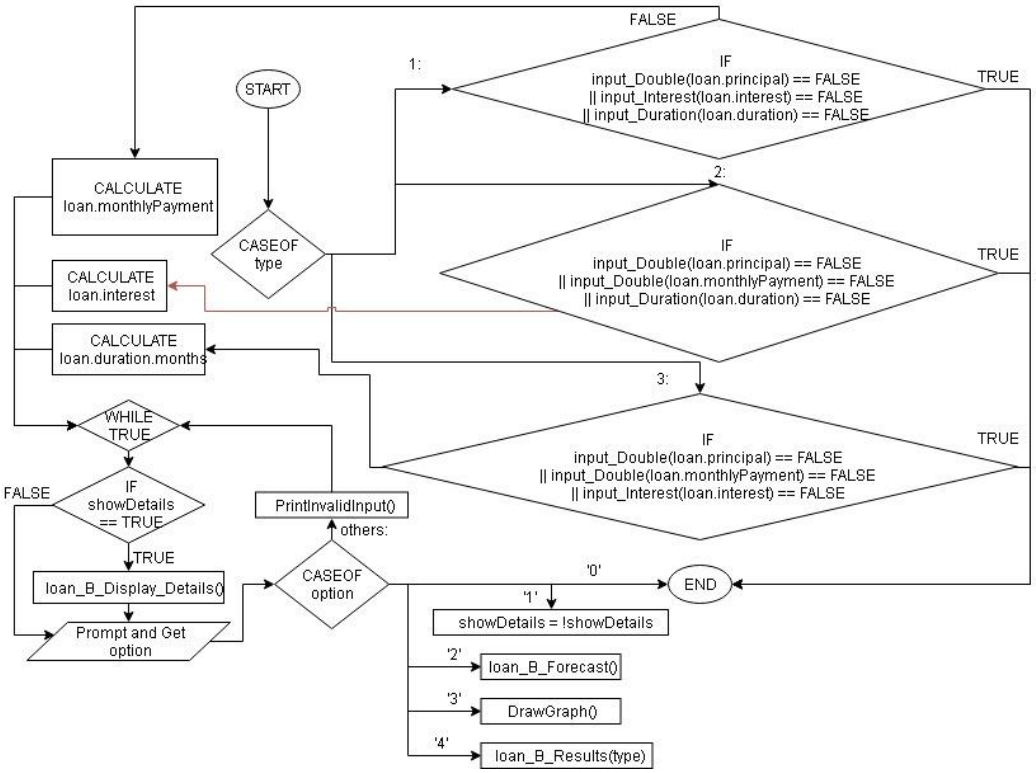



```
loan_B
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': loan_B_Results(1)
            '2': loan_B_Results(2)
            '3': loan_B_Results(3)
            '4': loan_B_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

i) **Results**

`void loan_B_Results(const int type)`

➢ Execute a range of things depending on the user's option from loan_B().




```
loan_B_Results
    CASEOF type
        1: IF   input_Double(loan.principal) == FALSE
               || input_Interest(loan.interest) == FALSE
               || input_Duration(loan.duration) == FALSE
           return
           ELSE
           CALCULATE loan.monthlyPayment
           ENDIF
        2: IF   input_Double(loan.principal) == FALSE
               || input_Double(loan.monthlyPayment) == FALSE
               || input_Duration(loan.duration) == FALSE
           return
           ELSE
           CALCULATE loan.interest
           ENDIF
        3: IF   input_Double(loan.principal) == FALSE
               || input_Double(loan.monthlyPayment) == FALSE
               || input_Interest(loan.interest) == FALSE
           return
           ELSE
           CALCULATE loan.duration.months
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
            loan_B_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': loan_B_Forecast()
            '3': DrawGraph()
            '4': loan_B_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# 2.3: Car Loan

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void loan_Car_Display_Details()`

➢ Display details regarding the loan.

b) **Forecast**

`void loan_Car_Forecast(char title[])`

➢ Displays a list of details for every single month of the loan.

c) **Compute Graph**

`double loan_Car_Compute_Graph(double months)`

➢ Used as a function pointer by the graphing function in DM.

d) **FAQ**

`void loan_Car_FAQ()`

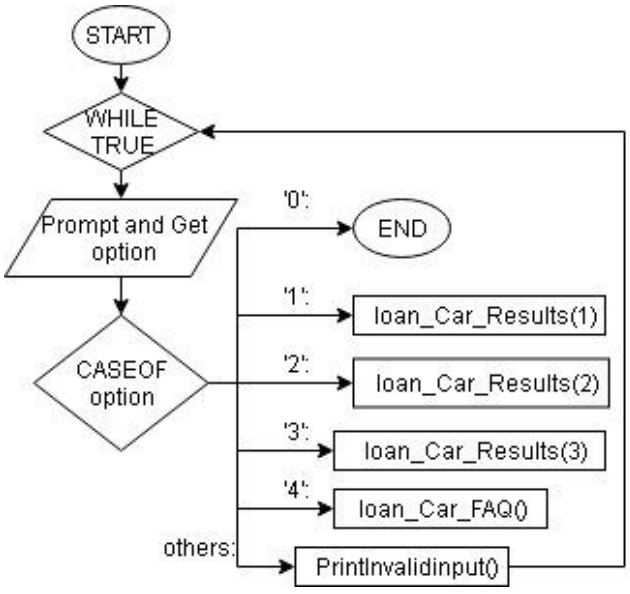
➢ Opens the FAQ menu.

---

e) **Loan Car**

`void loan_Car()`

➢ Displays the menu for this loan.

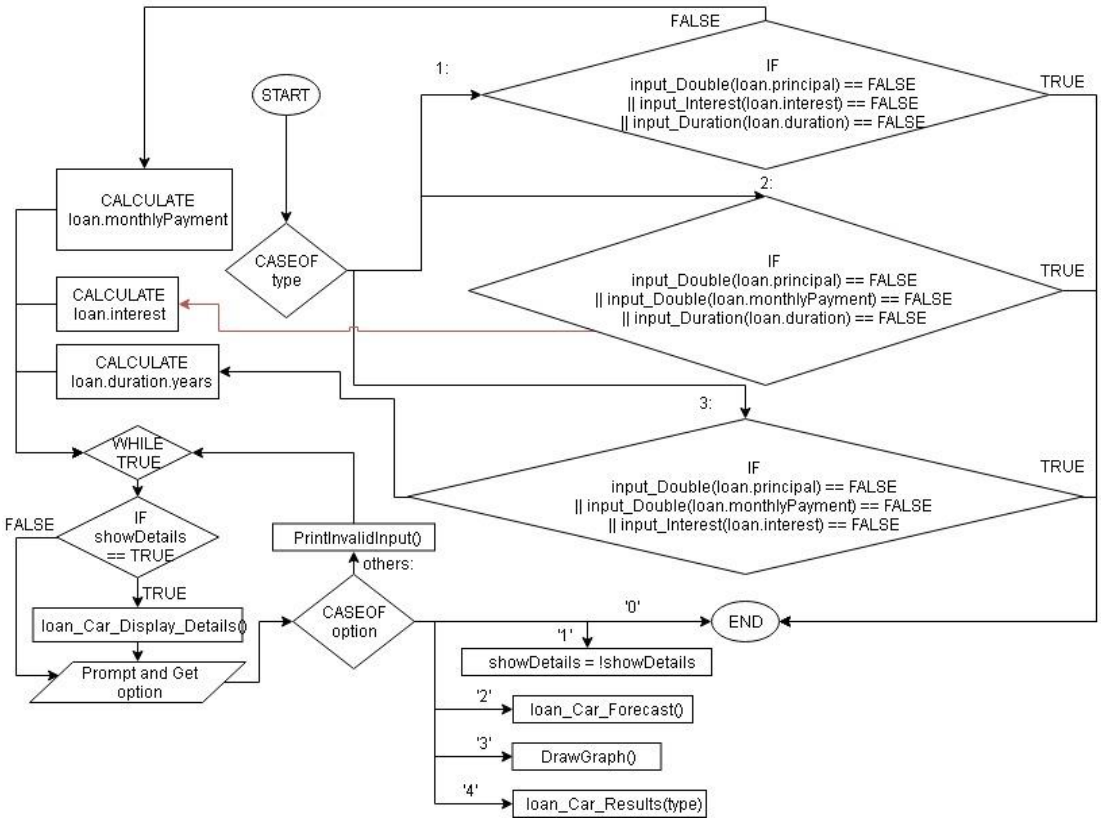



```
loan_Car
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': loan_Car_Results(1)
            '2': loan_Car_Results(2)
            '3': loan_Car_Results(3)
            '4': loan_Car_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

f) **Results**

`void loan_Car_Results(const int type)`

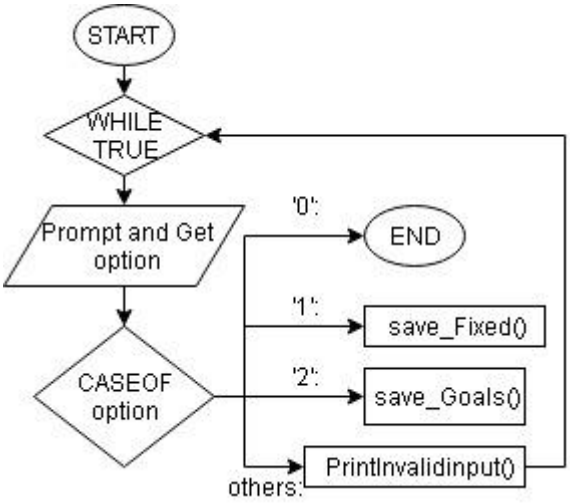
➢ Execute a range of things depending on the user's option from loan_Car().




```
loan_Car_Results
    CASEOF type
        1: IF   input_Double(loan.principal) == FALSE
               || input_Interest(loan.interest) == FALSE
               || input_Duration(loan.duration) == FALSE
            return
           ELSE
            CALCULATE loan.monthlyPayment
           ENDIF
        2: IF   input_Double(loan.principal) == FALSE
               || input_Double(loan.monthlyPayment) == FALSE
               || input_Duration(loan.duration) == FALSE
            return
           ELSE
            CALCULATE loan.interest
           ENDIF
        3: IF   input_Double(loan.principal) == FALSE
               || input_Double(loan.monthlyPayment) == FALSE
               || input_Interest(loan.interest) == FALSE
            return
           ELSE
            CALCULATE loan.duration.years
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
            loan_Car_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': loan_Car_Forecast()
            '3': DrawGraph()
            '4': loan_Car_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# Part 3: Savings

Some savings models are so similar to certain loans that it would be difficult to differentiate the two. The four financial aspects mentioned in the loan section also apply here. The OpenSavings() function opens the main menu for this section. The following are diagrams for the function:




```
Open_Savings
    DOWHILE TRUE
        Prompt and Get option
        CASEOF
            '0': return
            '1': save_Fixed()
            '2': save_Goals()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

A struct, Savings, contains the variables utilised by the functions in this section.

```
struct Savings {
    double principal;
    double interest;//Annual interest rate
    double monthlyDeposit;
    struct Duration duration;
    double total;//The target
    double actualTotal;
} savings; //A variable to store temporary values.
```

The variable, total, represents the target set by the user, whereas actualTotal stores the calculated net total after a certain duration.

# 3.1: Fixed Deposit

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void save_Fixed_Display_Details()`

➢ Display details regarding the savings.

b) **Forecast**

`void save_Fixed_Forecast(char title[])`

➢ Displays a list of details for every single month of the savings.

c) **Compute Graph**

`double save_Fixed_Compute_Graph(double months)`

➢ Used as a function pointer by the graphing function in DM.

d) **Compute Specific Interest**

`double save_Fixed_Compute_Specific_Interest(int iteration)`

➢ Computes the interest at a particular timeframe/iteration.

e) **FAQ**

`void save_Fixed_FAQ()`

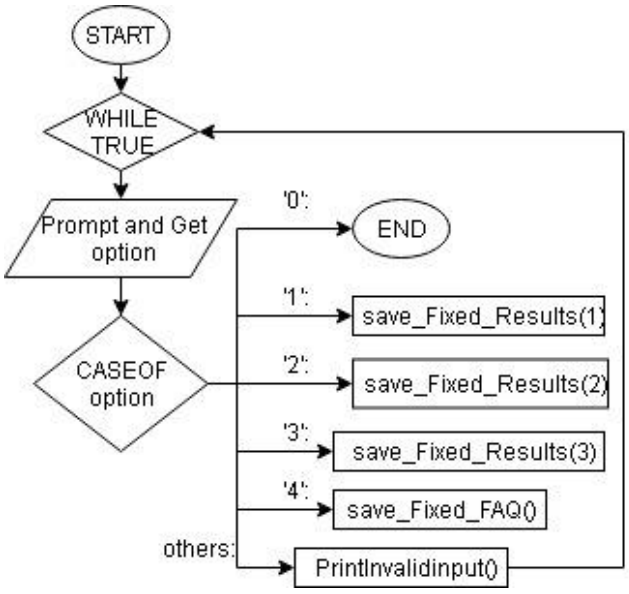
➢ Displays the FAQ menu.

---

f) **Save Fixed**

`void save_Fixed()`

➢ Displays the menu for this savings model.

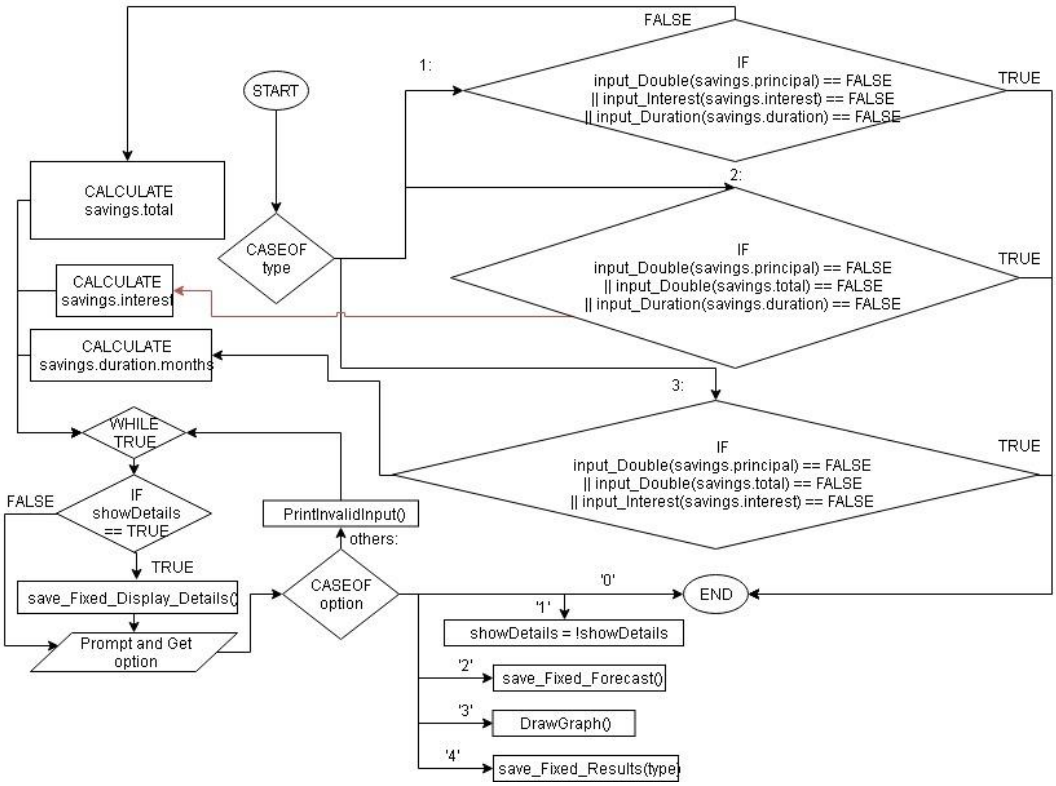



```
save_Fixed
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': save_Fixed_Results(1)
            '2': save_Fixed_Results(2)
            '3': save_Fixed_Results(3)
            '4': save_Fixed_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

g) **Results**

```
void save_Fixed_Results(const int type)
```

➢ Execute a range of things depending on the user's option from save_Fixed().




```
save_Fixed_Results
    CASEOF type
        1: IF   input_Double(savings.principal) == FALSE
               || input_Interest(savings.interest) == FALSE
               || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE savings.total
           ENDIF
        2: IF   input_Double(savings.principal) == FALSE
               || input_Double(savings.total) == FALSE
               || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE savings.interest
           ENDIF
        3: IF   input_Double(savings.principal) == FALSE
               || input_Double(savings.monthlyPayment) == FALSE
               || input_Interest(savings.interest) == FALSE

            return
           ELSE
            CALCULATE savings.duration.months
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
            save_Fixed_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': save_Fixed_Forecast()
            '3': DrawGraph()
            '4': save_Fixed_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# 3.2: Deposit Goals

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void save_Goals_Display_Details()`

➢ Display details regarding the savings.

b) **Forecast**

`void save_Goals_Forecast(char title[])`

➢ Displays a list of details for every single month of the savings.

c) **Compute Graph**

`double save_Goals_Compute_Graph(double months)`

➢ Used as a function pointer by the graphing function in DM.

d) **Compute Total**

`double save_Goals_Compute_Total(double principal, double annualInterest, double monthlyDeposit, int months)`

➢ Returns the total value as a double.

e) **Compute Monthly Deposit**

`double save_Goals_Compute_MonthlyDeposit(double principal, double total, double annualInterest, int months)`

➢ Returns the monthly deposit value as a double.

f) **Compute Interest Rate**

`double save_Goals_Compute_InterestRate(double principal, double total, double monthlyDeposit, int months, double start, int iterations)`

➢ Computes the interest rate using Newton's approximation.

g) **Compute Duration**

`int save_Goals_Compute_Duration(double principal, double total, double annualInterest, double monthlyDeposit)`

➢ Returns the duration value as a rounded-up integer.

h) **FAQ**

`void save_Goals_FAQ()`

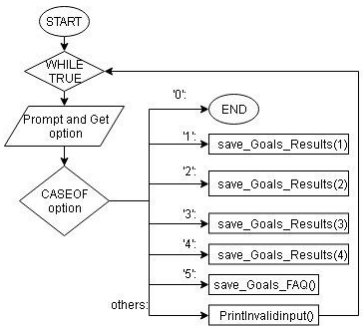
➢ Displays the FAQ menu.

---

i) **Save Goals**

`void save_Goals()`

➢ Displays the menu for this savings model.

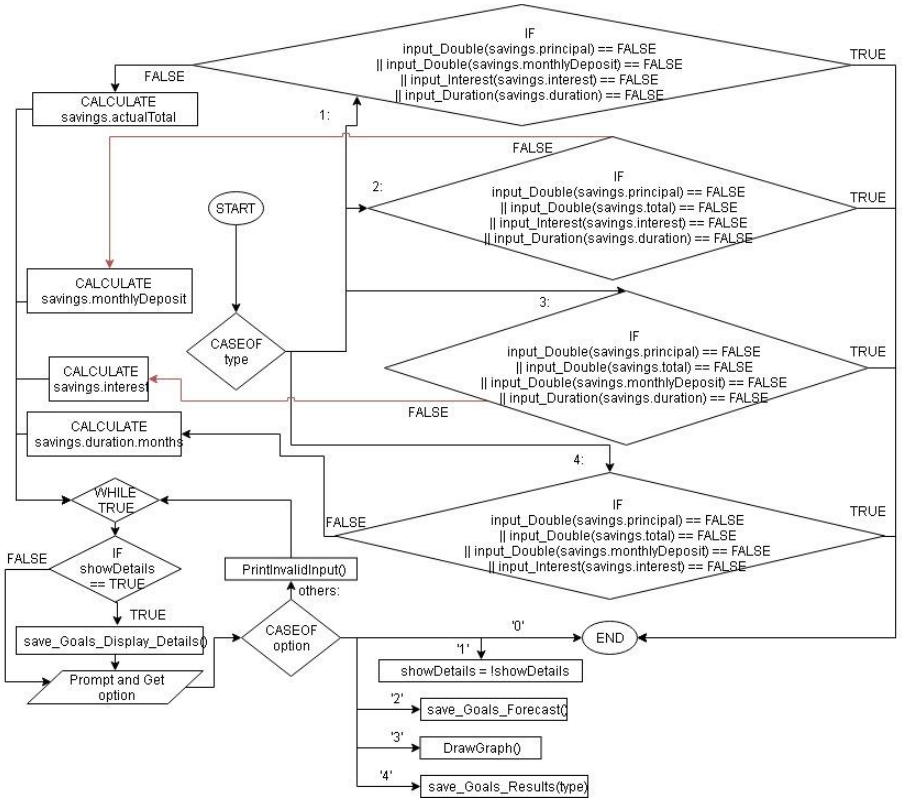



```
save_Goals
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': save_Goals_Results(1)
            '2': save_Goals_Results(2)
            '3': save_Goals_Results(3)
            '4': save_Goals_Goals(4)
            '5': save_Goals_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

j) **Results**

```
void save_Goals_Results(const int type)
```

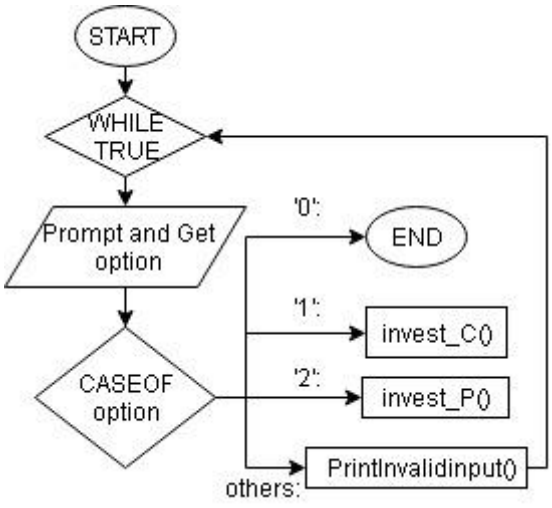
➢ Execute a range of things depending on the user's option from save_Goals().




```
save_Goals_Results
    CASEOF type
        1: IF   input_Double(savings.principal) == FALSE
                || input_Double(savings.monthlyDeposit) == FALSE
                || input_Interest(savings.interest) == FALSE
                || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE savings.actualTotal
           ENDIF
        2: IF   input_Double(savings.principal) == FALSE
                || input_Double(savings.total) == FALSE
                || input_Interest(savings.interest) == FALSE
                || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE savings.monthlyDeposit
           ENDIF
        3: IF   input_Double(savings.principal) == FALSE
                || input_Double(savings.total) == FALSE
                || input_Double(savings.monthlyDeposit) == FALSE
                || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE savings.interest
           ENDIF
        4: IF   input_Double(savings.principal) == FALSE
                || input_Double(savings.total) == FALSE
                || input_Double(savings.monthlyDeposit) == FALSE
                || input_Interest(savings.interest) == FALSE
            return
           ELSE
            CALCULATE savings.duration.months
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
            save_Goals_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': save_Goals_Forecast()
            '3': DrawGraph()
            '4': save_Goals_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# Part 4: Investments

Financial investments come in many forms, including businesses, goods, properties and services. For the sake of simplicity, we generalise and condense them into an elementary calculator which determines their net worth after a specified duration. The cause of their growth is approximated by a fixed interest rate. If the reader is familiar with various financial instruments or even just the program, he/she would discover that this calculator is almost identical to some of the others. To spice things up, we have included another feature whereby the user can estimate the NPV or the IRR of an investment. The four financial aspects mentioned in the loan section apply to the first calculator. The OpenInvest() function opens the main menu for this section. The following are diagrams for the function:




```
Open_Invest
    DOWHILE TRUE
        Prompt and Get option
        CASEOF
            '0': return
            '1': invest_C()
            '2': invest_P()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

```
struct Invest {
    double principal;
    double interest;//Annual interest rate
    struct Duration duration;
    double total;
} invest; //A variable to store temporary values.
//Linked list
struct Invest_P_List {
    struct Invest_P_List* next;
    double in;//Positive cash flow
};
struct Invest_P {
    double principal;
    double interest;
    double npv; //Normal Present Value
    double irr; //Interna Rate of Return
    struct Invest_P_List* list; // The first elemen
} investP; //A variable to store temporary values.
```

Invest_P_List is a linked list which is used for the second part (NPV and IRR).

# 4.1: CAGR

More info: Link 1, Link 2

Functions:

a) **Display Details**

`void invest_C_Display_Details()`

➢ Display details regarding the investment.

b) **Forecast**

`void invest_C_Forecast(char title[])`

➢ Displays a list of details for every single year of the investment.

c) **Compute Graph**

`double invest_C_Compute_Graph(double year)`

➢ Used as a function pointer by the graphing function in DM.

d) **FAQ**

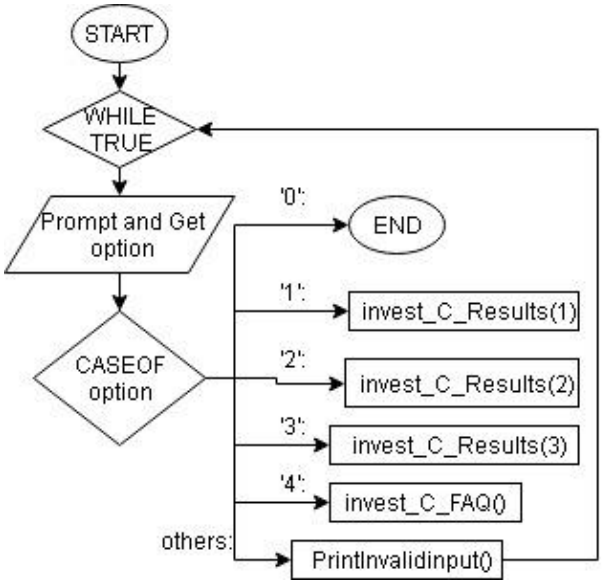
`void invest_C_FAQ()`

➢ Displays the FAQ menu.

e) **Invest C**

`void invest_C()`

➢ Displays the menu for this investment model.

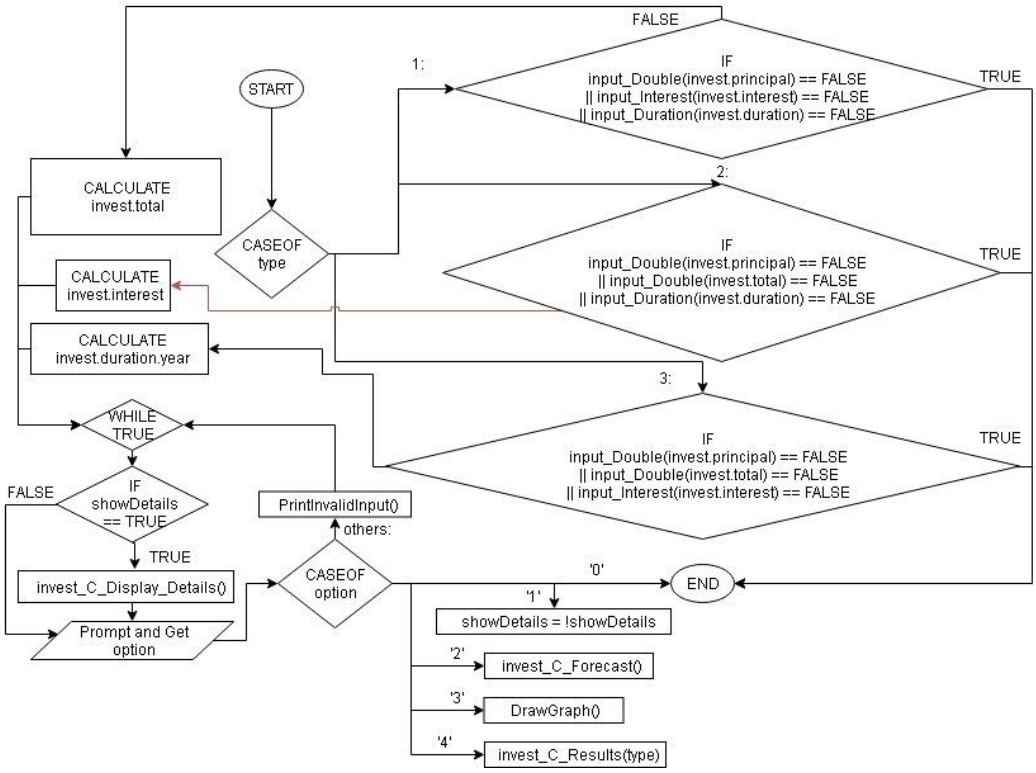



```
invest_C
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': invest_C_Results(1)
            '2': invest_C_Results(2)
            '3': invest_C_Results(3)
            '4': invest_C_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

f) **Results**

```
void save_Goals_Results(const int type)
```

➢ Execute a range of things depending on the user's option from invest_C().




```
invest_C_Results
    CASEOF type
        1: IF   input_Double(savings.principal) == FALSE
               || input_Interest(savings.interest) == FALSE
               || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE invest.total
           ENDIF
        2: IF   input_Double(savings.principal) == FALSE
               || input_Double(savings.total) == FALSE
               || input_Duration(savings.duration) == FALSE
            return
           ELSE
            CALCULATE invest.interest
           ENDIF
        3: IF   input_Double(savings.principal) == FALSE
               || input_Double(savings.total) == FALSE
               || input_Interest(savings.interest) == FALSE
            return
           ELSE
            CALCULATE invest.duration.years
           ENDIF
    ENDCASE
    DOWHILE TRUE
        IF showDetails == TRUE
            invest_C_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': invest_C_Forecast()
            '3': DrawGraph()
            '4': invest_C_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# 4.2: Prospects

More info: NPV, IRR, Math

Functions:

a) **Display Details**

`void invest_P_Display_Details()`

➢ Display details regarding the investment.

b) **FAQ**

`void invest_P_FAQ()`

➢ Displays the FAQ menu.

c) **Compute NPV**

`double invest_P_Compute_NPV(struct Invest_P_List* head, double principal, double interest)`

➢ Returns the NPV as a double

d) **Compute IRR**

`double invest_P_Compute_IRR(struct Invest_P_List* head, double principal)`

➢ Computes the IRR with an error margin of 0.01.

e) **Add To List**

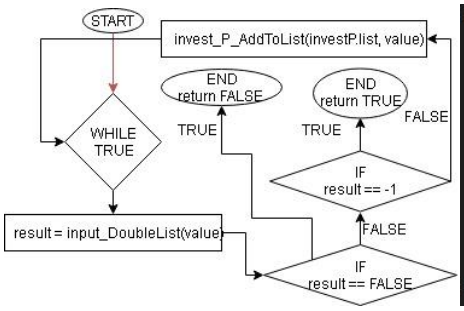
`void invest_P_AddToList(struct Invest_P_List* head, double in)`

➢ Adds a double to the linked list.

f) **Input List**

`int invest_P_InputList(char title[])`

➢ Asks the user to specify the payment details. There is no limit to the number of payments/length of the list.

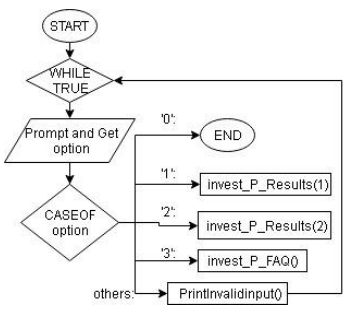



```
Input_List
    DOWHILE
        result = input_DoubleList(value)
        IF result == FALSE
            return FALSE
        ELSEIF result == -1
            return TRUE
        ELSE
            invest_P_AddToList(investP.list, value)
        ENDIF
    ENDDO
END
```

g) **Invest P**

`void invest_P()`

➢ Displays the menu for this investment model.

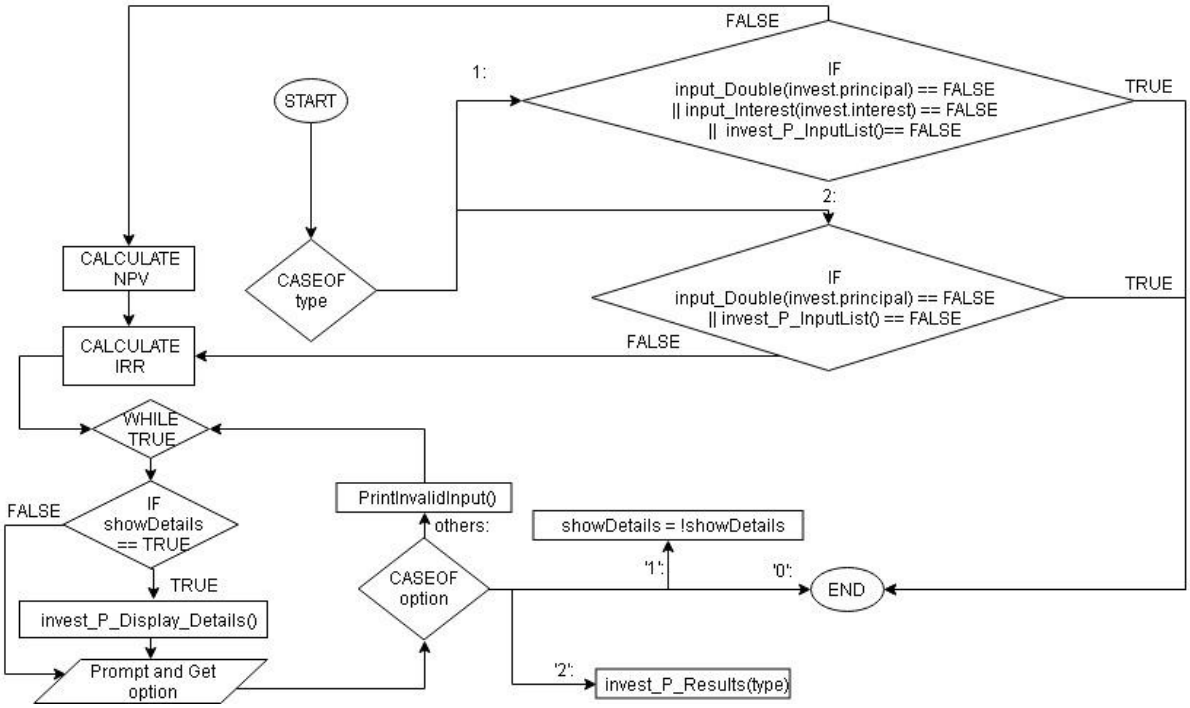



```
invest_P
    DOWHILE TRUE
        Prompt and Get option
        CASEOF option
            '0': return
            '1': invest_P_Results(1)
            '2': invest_P_Results(2)
            '3': invest_P_FAQ()
            others: PrintInvalidInput()
        ENDCASE
    ENDDO
END
```

h) **Results**

```
void save_Goals_Results(const int type)
```

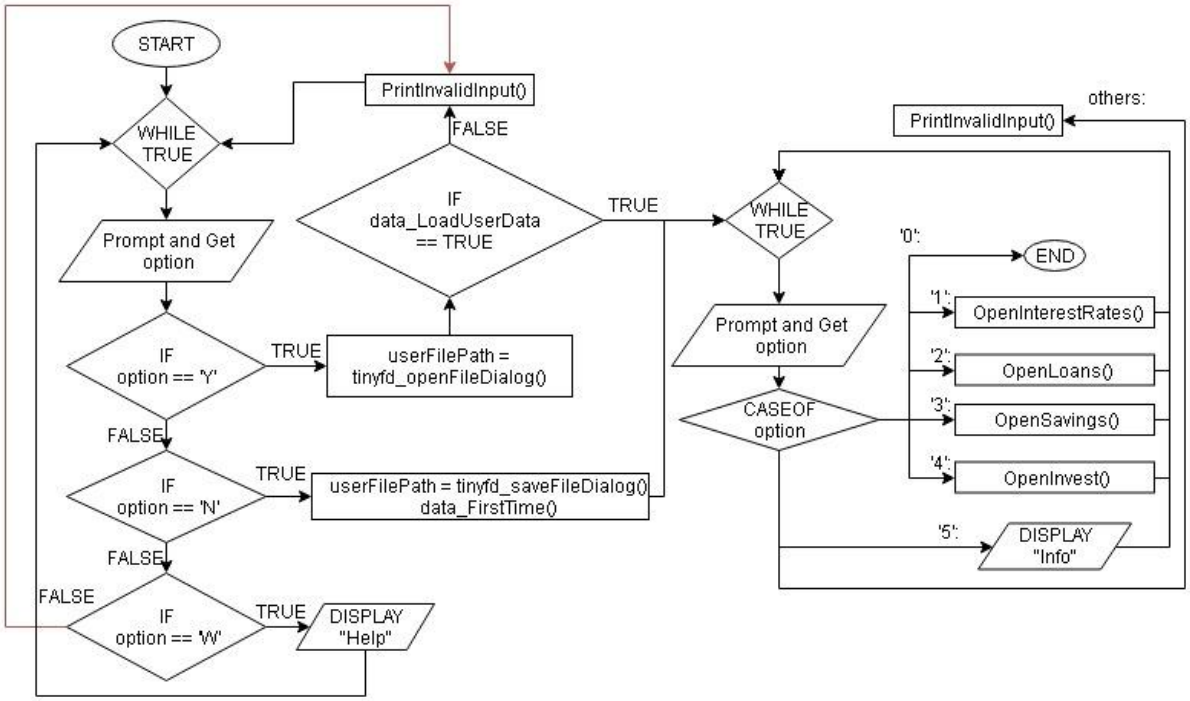
➢ Execute a range of things depending on the user's option from invest_P().




```
invest_P_Results
    CASEOF type
        1: IF   input_Double(invest.principal) == FALSE
               || input_Interest(invest.interest) == FALSE
               || invest_P_InputList()== FALSE
            return
           ELSE
            CALCULATE invest.npv
           ENDIF
        2: IF   input_Double(invest.principal) == FALSE
               || invest_P_InputList()== FALSE
            return
           ENDIF
    ENDCASE
    CALCULATE invest.irr
    DOWHILE TRUE
        IF showDetails == TRUE
            invest_P_DisplayDetails()
        ENDIF
        Prompt and Get option
        CASEOF option
            '0': return
            '1': showDetails = !showDetails
            '2': invest_P_Results(type)
            others:
        ENDCASE
    ENDDO
END
```

# Part 5: Main

The main function is the first to be called upon program execution. Now that we have covered almost every area of the program, we can now piece together a coherent framework in the function.




```
main
DOWHILE TRUE
    Prompt and Get option
    IF option == 'Y'
        userFilePath = tinyfd_openFileDialog()
        IF  data_LoadUserData == TRUE
            break
        ENDIF
        PrintInvalidInput()
    ELSEIF option == 'N'
        userFilePath = tinyfd_saveFileDialog()
        data_FirstTime()
        break
    ELSEIF option == 'W'
        DISPLAY "Help"
    ELSE
        PrintInvalidInput()
    ENDIF
ENDDO
DOWHILE TRUE
    Prompt and Get option
    CASEOF option
        '0': return
        '1': OpenInterestRates()
        '2': OpenLoans()
        '3': OpenSavings
        '4': OpenInvest()
        '5': DISPLAY "Info"
        others: PrintInvalidInput()
    ENDCASE
ENDO
END
```

At the beginning of the function, the user is prompted where to load a profile or to create a new one. If the user chooses the former, tinyfd_openFileDialog() from the “tinyfiledialogs” library is called to open a file dialog for the chooser to select the file. To create a new file, tinyfd_saveFileDialog() is called instead for the user to select the folder to save the file in.

# Part 6: IPO Charts

## 6.1: File Manager

### a) Read File

| Input | Process | Output |
|---|---|---|
| file<br>closeStream | READ file | text |

### b) Write File

| Input | Process | Output |
|---|---|---|
| file<br>contents<br>closeStream | WRITE contents TO file | void |

# 6.2: Custom Printf

### a) Periodic Printf

| Input | Process | Output |
|---|---|---|
| text<br>interval | 1. DISPLAY text<br>2. DELAY interval | void |

### b) Print Loop

| Input | Process | Output |
|---|---|---|
| text<br>repetitions<br>newLine | 1. i = 0<br>2. REPEAT<br>DISPLAY text<br>UNTIL I >= repetitions | void |

### c) Print Invalid Input

| Input | Process | Output |
|---|---|---|
| void | DISPLAY “Invalid” | void |

### d) Print Invalid Negative

| Input | Process | Output |
|---|---|---|
| void | DISPLAY “Invalid” | void |

## 6.3: Data Manager

### a) Is Empty

| Input | Process | Output |
|---|---|---|
| text | CHECK IF text IS EMPTY | TRUE/FALSE |

### b) Is Integer

| Input | Process | Output |
|---|---|---|
| text | CHECK IF text IS INTEGER | TRUE/FALSE |

### c) Exceed Char Limit

| Input | Process | Output |
|---|---|---|
| text<br>delimiter<br>max | CHECK IF NUMBER OF delimiter IN text > max | TRUE/FALSE |

### d) Is Double

| Input | Process | Output |
|---|---|---|
| text | CHECK IF text IS DOUBLE | TRUE/FALSE |

### e) Is Date

| Input | Process | Output |
|---|---|---|
| text | CHECK IF text IS DATE | TRUE/FALSE |

### f) Split String

| Input | Process | Output |
|---|---|---|
| destination<br>source<br>delimiter | 1. SPLIT source BY delimiter<br>2. STORE IN destination | destination |

### g) Int To Months

| Input | Process | Output |
|---|---|---|
| number | CONVERT number TO month | month |

### h) Get Current Date

| Input | Process | Output |
|---|---|---|
| void | GET currentDate | currentDate |

### i) Int To String

| Input | Process | Output |
|---|---|---|
| number | CONVERT number TO string | string |

### j) OPEN FAQ

| Input | Process | Output |
|---|---|---|
| type | OPEN FAQ | void |

### k) Draw Graph

| Input | Process | Output |
|---|---|---|
| title<br>function<br>xMax<br>xTitle<br>yOffset<br>yMax<br>yTitle | 1. DISPLAY title<br>2. CALCULATE FROM function<br>3. DISPLAY GRAPH | void |

### l) Update User Data

| Input | Process | Output |
|---|---|---|
| reset | WRITE JSON TO file | void |

### m) First Time

| Input | Process | Output |
|---|---|---|
| void | 1. PROMPT AND GET name<br>2. PROMPT AND GET age | void |

### n) Load User Data

| Input | Process | Output |
|---|---|---|
| void | PARSE JSON | void |

## 6.4: Input Manager

**a) Input String**

| Input | Process | Output |
|---|---|---|
| title<br>query<br>ptr | 1. DISPLAY title, query<br>2. PROMPT AND GET ptr | TRUE/FALSE |

**b) Input Int**

| Input | Process | Output |
|---|---|---|
| title<br>query<br>ptr<br>positiveValues | 1. DISPLAY title, query<br>2. PROMPT AND GET ptr | TRUE/FALSE |

**c) Input Double**

| Input | Process | Output |
|---|---|---|
| title<br>query<br>ptr<br>positiveValues | 1. DISPLAY title, query<br>2. PROMPT AND GET ptr | TRUE/FALSE |

**d) Input Double List**

| Input | Process | Output |
|---|---|---|
| title<br>query<br>ptr<br>positiveValues | 1. DISPLAY title, query<br>2. PROMPT AND GET ptr | TRUE/FALSE |

**e) Input Interest**

| Input | Process | Output |
|---|---|---|
| title<br>ptr | 1. DISPLAY title<br>2. PROMPT AND GET ptr | TRUE/FALSE |

### f) Input Date Duration

| Input | Process | Output |
|---|---|---|
| month1<br>year1<br>month2<br>year2 | CALCULATE duration | duration |

### g) Input Duration

| Input | Process | Output |
|---|---|---|
| title<br>duration | Prompt AND Get duration | TRUE/FALSE |

## 6.5: Interest Rates

### a) Reset Rates

| Input | Process | Output |
|---|---|---|
| index | 1. interestRates[index].name = “”<br>2. interestRates[index].rate1 = 0<br>3. interestRates[index].rate2 = 0<br>4. interestRates[index].rate3 = 0 | void |

### b) Display Length

| Input | Process | Output |
|---|---|---|
| void | CALCULATE length | length |

### c) Is Full

| Input | Process | Output |
|---|---|---|
| void | CHECK IF interestRates IS FULL | TRUE/FALSE |

### d) Update Data Elements

| Input | Process | Output |
|---|---|---|
| index | rateElements = interestRates[index] | void |

### e) Display Element Index

| Input | Process | Output |
|---|---|---|
| index | DISPLAY interestRates[index] | void |

### f) Display Element

| Input | Process | Output |
|---|---|---|
| index | DISPLAY interestRates[index] | void |

### g) Display

| Input | Process | Output |
|---|---|---|
| void | DISPLAY interestRates | void |

### h) Open Interest Rates

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### i) Add

| Input | Process | Output |
|---|---|---|
| void | 1. PROMPT AND GET interestRate<br>2. ADD interestRate TO interestRates | void |

### j) Select Index

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. GET index FROM interestRates | index |

### k) Select Value

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. rates_Select_Index()<br>3. PROMPT AND GET index | index |

### l) Modify

| Input | Process | Output |
|---|---|---|
| void | 1. index = rates_Select_Index()<br>2. PROMPT AND GET interestRates[index] | void |

### m) Delete

| Input | Process | Output |
|---|---|---|
| void | 1. index = rates_Select_Index()<br>2. DELETE interestRates[index] | void |

### n) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

# 6.6: Interest Only

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY loan | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### e) Loan I

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### f) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, duration<br>CALCULATE total<br>2: PROMPT AND GET principal, total, duration<br>CALCULATE interest<br>3: PROMPT AND GET principal, total, interest<br>CALCULATE months<br>ENDCASE | void |

# 6.7: Interest & Monthly Payments

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY loan | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) Compute Monthly Payment

| Input | Process | Output |
|---|---|---|
| principal<br>annualInterest<br>months | CALCULATE monthlyPayment | monthlyPayment |

### e) Compute Interest Rate

| Input | Process | Output |
|---|---|---|
| principal<br>months<br>monthlyPayment<br>start<br>iterations | CALCULATE interest | interest |

### f) Compute Duration

| Input | Process | Output |
|---|---|---|
| principal<br>monthlyPayment<br>annualInterest | CALCULATE months | months |

### g) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### h) Loan B

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### i) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, duration<br>CALCULATE monthlyPayment<br>2: PROMPT AND GET principal, monthlyPayment, duration<br>CALCULATE interest<br>3: PROMPT AND GET principal, monthlyPayment, interest<br>CALCULATE months<br>ENDCASE | void |

# 6.8: Car Loan

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY loan | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### e) Loan Car

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### f) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, duration<br>CALCULATE monthlyPayment<br>2: PROMPT AND GET principal, monthlyPayment, duration<br>CALCULATE interest<br>3: PROMPT AND GET principal, monthlyPayment, interest<br>CALCULATE years<br>ENDCASE | void |

# 6.9: Fixed Deposit

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY savings | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) Compute Specific Interest

| Input | Process | Output |
|---|---|---|
| iteration | CALCULATE interest | interest |

### e) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### f) Save Fixed

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### g) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, duration<br>CALCULATE total<br>2: PROMPT AND GET principal, total, duration<br>CALCULATE interest<br>3: PROMPT AND GET principal, total, interest<br>CALCULATE months<br>ENDCASE | void |

# 6.10: Deposit Goals

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY savings | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) Compute Total

| Input | Process | Output |
|---|---|---|
| principal<br>annualInterest<br>monthlyPayment<br>months | CALCULATE actualTotal | total |

### e) Compute Monthly Deposit

| Input | Process | Output |
|---|---|---|
| principal<br>total<br>annualInterest<br>months | CALCULATE monthlyDeposit | monthlyDeposit |

### f) Compute Interest Rate

| Input | Process | Output |
|---|---|---|
| principal<br>total<br>monthlyDeposit<br>months<br>start<br>iterations | CALCULATE interest | interest |

### g) Compute Duration

| Input | Process | Output |
|---|---|---|
| principal<br>total<br>annualInterest<br>monthlyDeposit | CALCULATE months | months |

### h) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### i) Save_Goals

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### j) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, monthlyDeposit, interest, duration<br>CALCULATE actualTotal<br>2: PROMPT AND GET principal, total, interest, duration<br>CALCULATE monthlyDeposit<br>3: PROMPT AND GET principal, total, monthlyDeposit, duration<br>CALCULATE interest<br>4: PROMPT AND GET principal, total, monthlyDeposit, interest<br>CALCULATE months<br>ENDCASE | void |

# 6.11: CAGR

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) Forecast

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DISPLAY invest | void |

### c) Compute Graph

| Input | Process | Output |
|---|---|---|
| months | CALCULATE stuff | stuff |

### d) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### e) Invest C

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### f) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, duration<br>CALCULATE total<br>2: PROMPT AND GET principal, total, duration<br>CALCULATE interest<br>3: PROMPT AND GET principal, total, interest<br>CALCULATE years<br>ENDCASE | void |

# 6.12: Prospects

### a) Display Details

| Input | Process | Output |
|---|---|---|
| void | DISPLAY DETAILS | void |

### b) FAQ

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### c) Compute NPV

| Input | Process | Output |
|---|---|---|
| payments<br>principal<br>interest | CALCULATE npv | npv |

### d) Compute IRR

| Input | Process | Output |
|---|---|---|
| payments<br>principal | CALCULATE irr | irr |

### e) Add To List

| Input | Process | Output |
|---|---|---|
| list<br>in | ADD in TO list | void |

### f) Input List

| Input | Process | Output |
|---|---|---|
| title | 1. DISPLAY title<br>2. DOWHILE TRUE<br>PROMPT AND GET in<br>Invest_P_AddToList(payments, in)<br>ENDDO | TRUE/FALSE |

### g) Invest P

| Input | Process | Output |
|---|---|---|
| void | DISPLAY MENU | void |

### h) Results

| Input | Process | Output |
|---|---|---|
| type | CASEOF type<br>1: PROMPT AND GET principal, interest, payments<br>CALCULATE npv<br>2: PROMPT AND GET principal, payments<br>CALCULATE irr<br>ENDCASE | void |

## 6.13: Main

### a) Main

| Input | Process | Output |
|---|---|---|
| void | 1. PROMPT AND GET loadFile<br>2. IF loadFile == TRUE<br>tinyfd_openFileDialog()<br>data_LoadUserData()<br>ELSE<br>tinyfd_saveFileDialog()<br>data_FirstTime()<br>ENDIF<br>3. PROMPT AND GET option<br>4. CASEOF option<br>'0': return<br>'1': OpenInterestRates()<br>'2': OpenLoans()<br>'3': OpenSavings<br>'4': OpenInvest()<br>'5': DISPLAY "Info"<br>others: PrintInvalidInput()<br>ENDCASE | void |

# Part 7: THE END