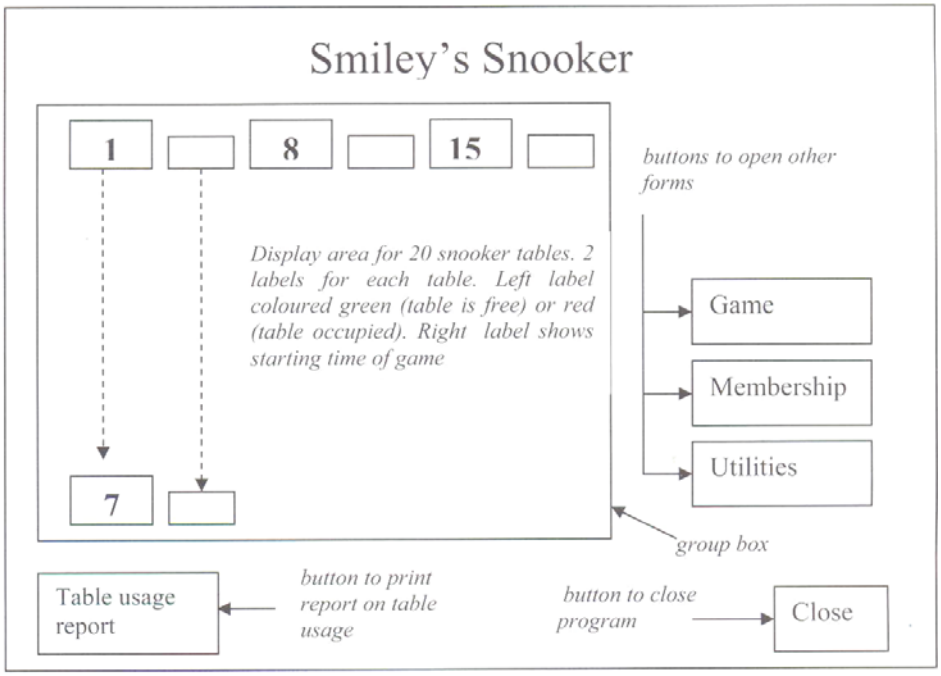
Design

General name	*Purpose*
Main form	Displays the 20 snooker tables and their state of play. Prints a report on daily usage of tables. Allows the other 3 forms to be loaded
Game form	Input / output appropriate data when a game either starts or finishes. Calculates the cost of a finished game. Stores details of games on file.
Members form	Allows new members to be added and others to be deleted. Displays full list of members. Prints a membership report.
Utilities form	Creates lost or missing files. Backs up files. Changes price of snooker game.

Main form

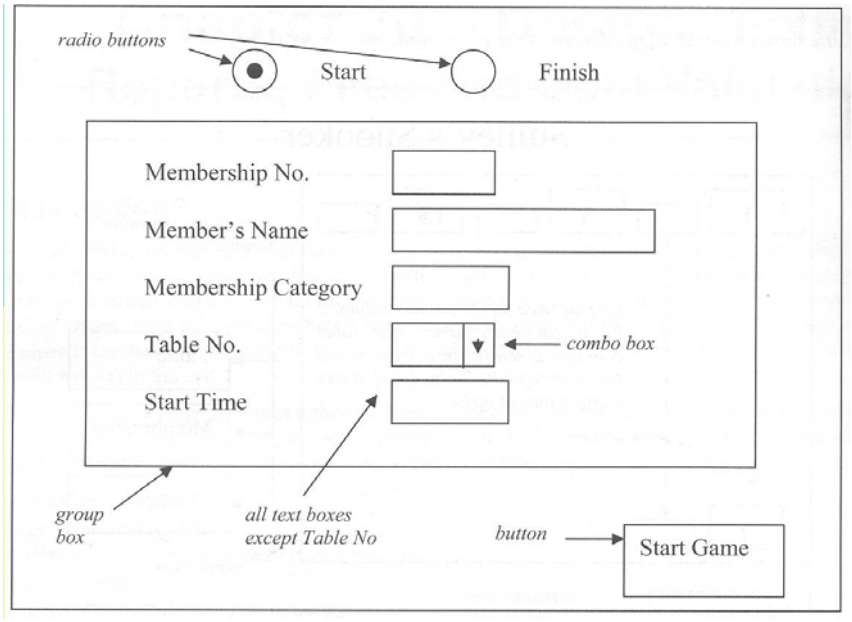
Sketch of the design of the main form.



Game form
Depending on which radio button is clicked, Start Game or Finish Game, a different set of controls is made visible. When the form first loads, by clicking the Game button on the Main form, the controls for starting a game will be visible.

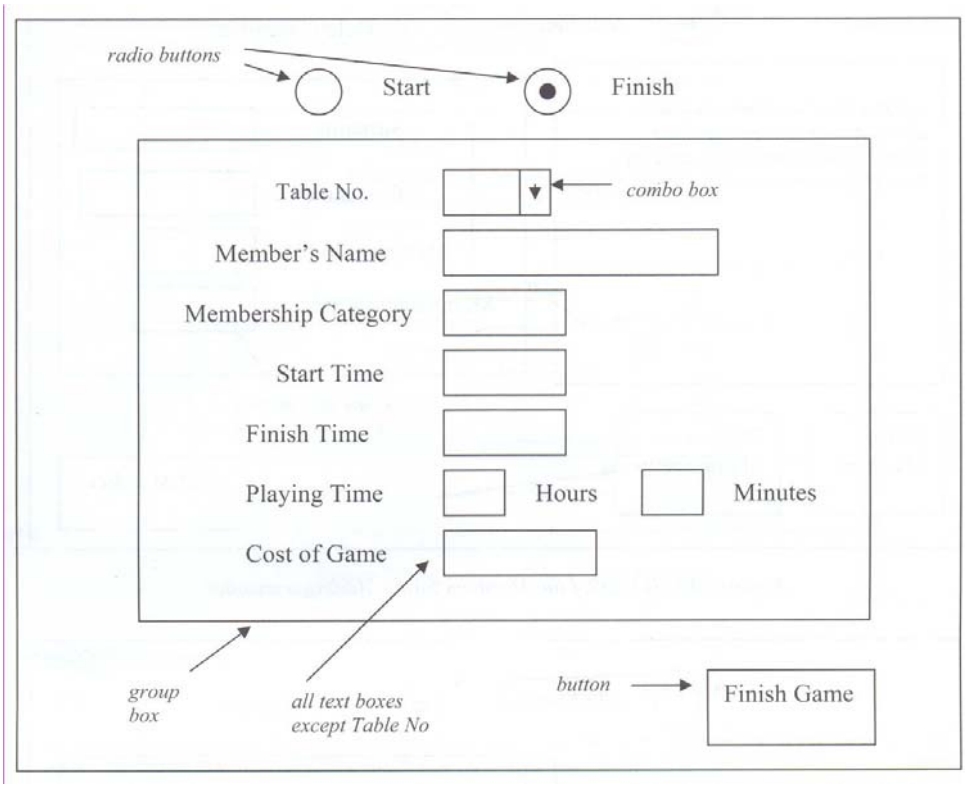
Starting a game
The following will happen:

- The receptionist enters the person's membership number.
- The member's name and their category of membership will automatically appear. The receptionist confirms these details with the player.
- The combo box will display the numbers of all those tables which are available for play. When the receptionist selects a table number the current time is taken from the system clock and displayed in the Start Time control. The receptionist will confirm with the member the starting time of the game for payment calculations.
- Clicking the Start Game button stores details of the game on a file and clears all the text boxes. It also changes the table's colour from green to red on the Main form and displays the starting time next to it.



Finishing a game
The following will happen:

- The receptionist will ask the player which table they have been using and selects this from the tables currently in use that are displayed in the combo box.
- Selecting a table number triggers the display of data in all the other controls.
- The finishing time of the game is taken from the system clock and the playing time and cost of the game calculated automatically.
- Clicking the Finish Game button stores details of the completed game on file and clears all the text boxes. It also changes the table's colour back to green and removes the start time next to it on the Main form.



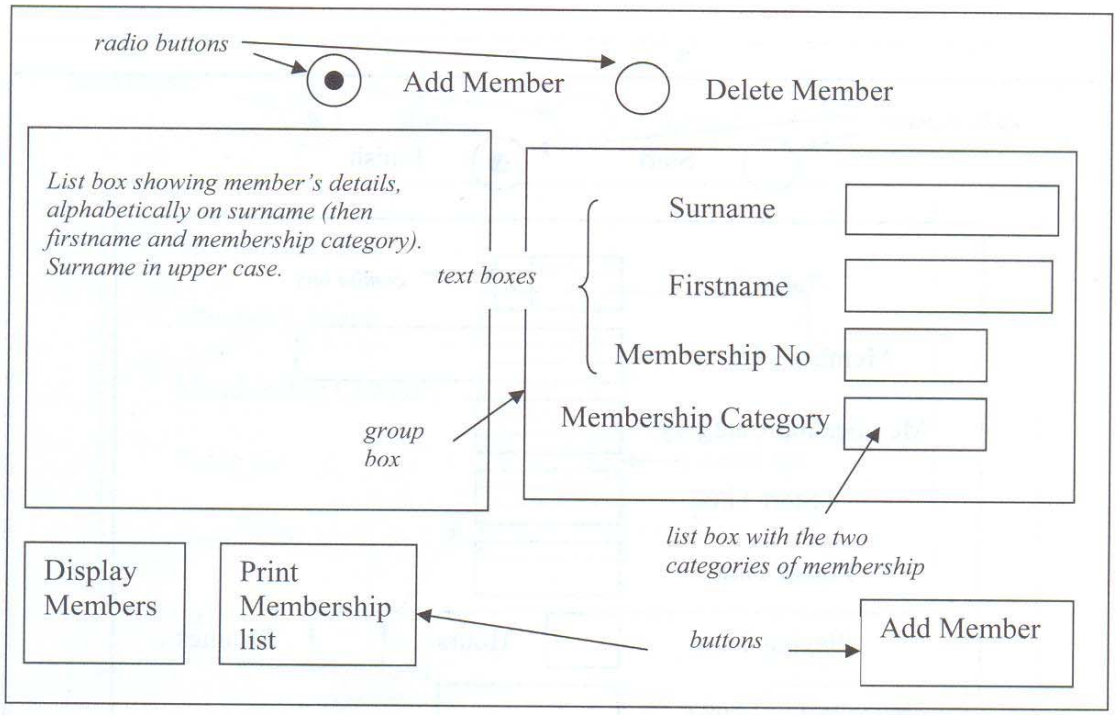
Members form

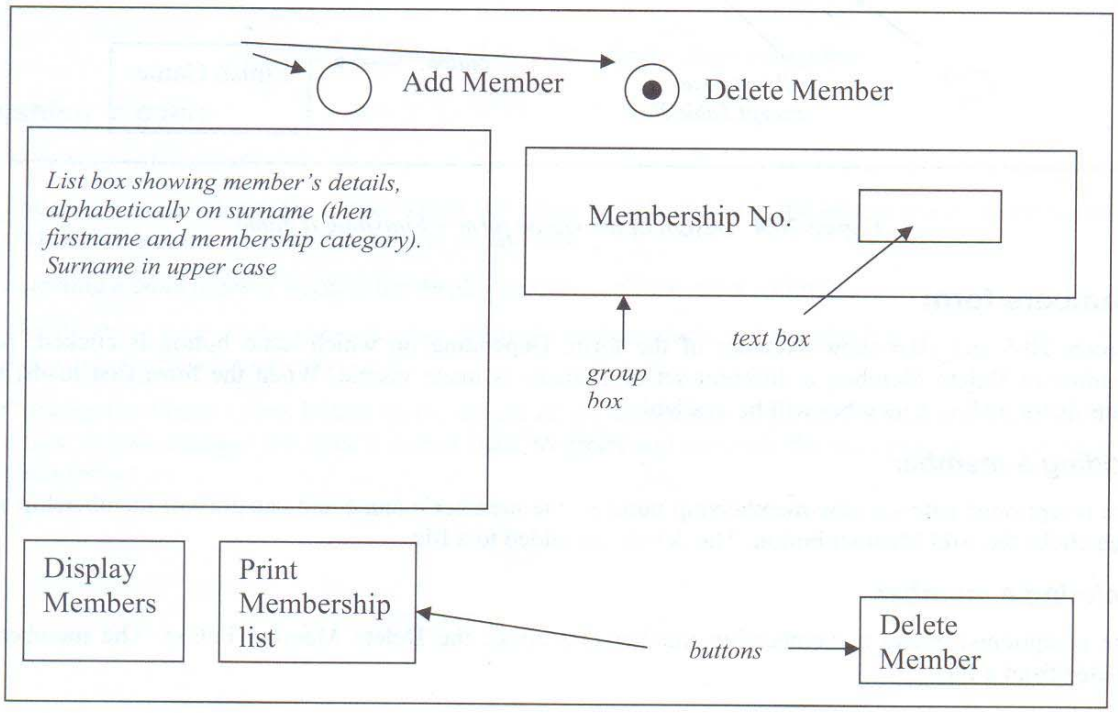
Depending on which radio button is clicked, Add Member or Delete Member, a different set of controls is made visible. When the form first loads, the controls for adding a member will be available.

Adding a member
The receptionist enters a new membership number, the member's name and category of membership and then clicks the Add Member button. The details are added to a file.

Deleting a member
The receptionist enters a membership number and clicks the Delete Member button. The member is deleted from a file.

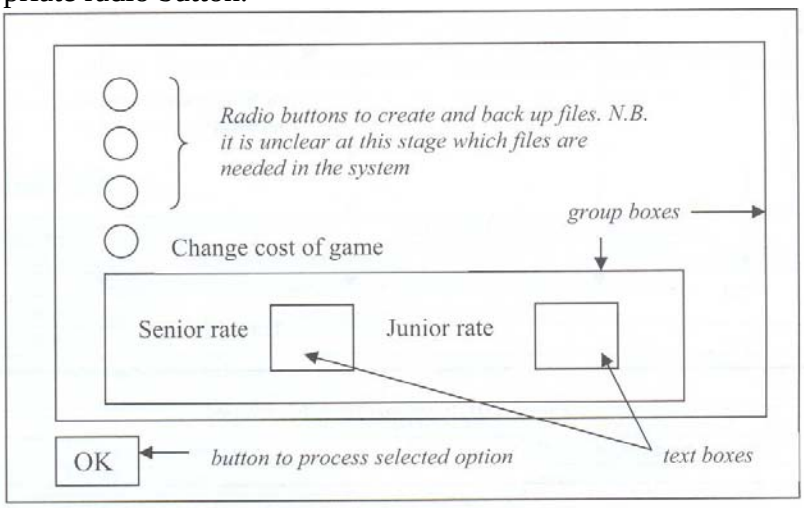
The list box displays details of each member. These details can be printed out.


radio buttons
Add Member
Delete Member
List box showing member's details, alphabetically on surname (then firstname and membership category). Surname in upper case.
text boxes
Surname
Firstname
Membership No
Membership Category
group box
list box with the two categories of membership
Display Members
Print Membership list
buttons
Add Member


Add Member
Delete Member
List box showing member's details, alphabetically on surname (then firstname and membership category). Surname in upper case
Membership No.
text box
group box
Display Members
Print Membership list
buttons
Delete Member

Utilities form

The controls for entering new payment rates for a game are only made visible if the user clicks the appropriate radio button.



Printed reports

My analysis found that Smiley's wants two printed reports:

- At any time during the day, but most likely at the end of the day, a report showing the usage of each table. This includes the total length of time the table has been in use, how many games were played on the table and how much income the table made. The overall income for the day is also required.
- An alphabetical list of members showing their name and category of membership, the total number of senior and junior members and the overall total.

.

Files

Four files are needed for Smiley's Snooker. Three of these will store records and therefore be random access files (for which I'll use a '.dat' extension in the file name) and one is a text file (' .txt' extension).

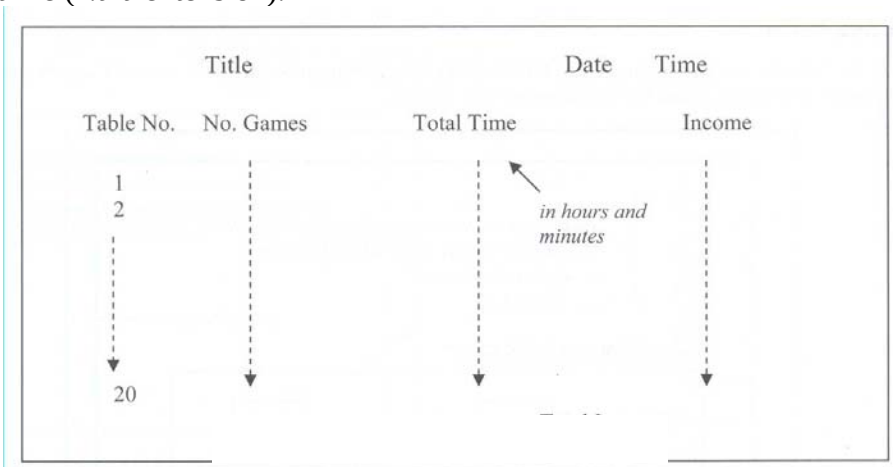
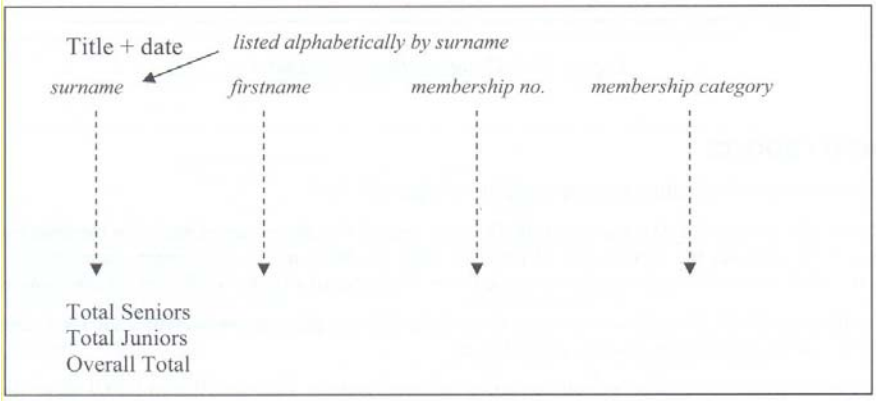


Figure 20.8: Report on table usage



CurrentGames.dat

Stores details of the status of each table (in use or not) and the membership number of the player responsible for the game. Note that the data in this file could be stored only in RAM, but if the user accidentally closed the program these details would be lost.

FinishedGames.dat

Stores details of games that have finished. It will be used for printing the report Smiley's wants on table usage. Smiley's does not require any details about which members used the tables in this report so these can be omitted. I do need to store details of the length of each game because the report only needs to show the total amount of time each table has been used. Because there are different rates for seniors and juniors I must either store the cost of the game or the membership category of the member in charge of the game. I have decided to store the cost.

Costs.txt

Stores the senior and junior rates as pence per minute.

Members.dat
Stores details of each member - membership number, name and category of membership. As explained below it also has a flagged field to indicate if the member has been deleted.

CurrentGames.dat file

External file name	CurrentGames.dat	**General name**	Current Games
Description	Stores details of the current status of each table - if in use details of the game		
Used for	Colouring the tables on the Main form green or red when the form loads Retrieving details of game when processing: a finished game		
Organisation	Random access		
Processing	File is created with all records initialised to appropriate values. TableID values correspond to record numbers and are used for direct access		

Record structure			
Field name	**Field description**	**Data type (and length)**	**No. bytes**
TableD	Table number (1 to 20)	Short	2
MemberID	Membership ID (2 letters + 4 numeric digits)	String(6)	6
StartTime	If table has a game, time game started	Date	8
Occupied	Whether a table has a game. Stores Y or N	Char	2
Record size	18 bytes		
Typical size of file	File always has 20 records (1 per table) – size is 360 bytes		

FinishedGames.dat file

External file name	FinishedGames.dat	**General name**	Finished Games
Description	Stores details of all completed games for the current day		
Used for	Producing: the printed report on daily table usage		
Organisation	Random access		
Processing	New records appended. No deletions / changes to data required. Linear searching of file to produce printed report on daily table usage.		

Record structure			
Field name	**Field description**	**Data type (and length)**	**No. bytes**
TableID	Table number (1 to 20)	Short	2
StartTime	Time game started	Date	8
FinishTime	Time game finished	Date	8
Cost	Cost of game	Decimal	16
Record size	34 bytes		
Typical size of file	1 record for each completed game. Assuming all tables are used all day (12 hours) and that a game averages 1 hour, 12 x 20 records stored - maximum total size 8160 bytes		

Costs.txt file

External file name	Costs.txt	**General name**	Costs
Description	Stores the senior and junior rates per minute for a game.		
Used for	Calculating cost of a finished game		
Organisation	Text file		
Structure of data	Only 1 line containing the two rates (e.g. 4, 3.5 etc)		
Typical size of file	A few bytes		

Members.dat file

The only things Smiley's needs to store about a member is their membership number, name and category of membership.

To delete a member I will flag one of its fields to indicate the record has been deleted. The record will still be there physically.

This means that I need a field to flag whether the record has been deleted. It will hold a single character Y or N.

.

To add a new record to the Members file I will overwrite the first record that has been flagged as deleted.

External file name	Members.dat	**General name**	Members
Description	Stores details of all current members		
Used for	Displaying data on Game form when a new game starts or a game finishes. Displaying list of members in Members form Producing printed list of members		
Organisation	Random access		
Processing	Records are logically deleted. Linear search made to find first logically deleted record when adding a new member - if none found record is appended.		
Record structure			
Field name	**Field description**	**Data type (and length)**	**No. bytes**
MemberID	Membership ID (2 letters + 4 numeric digits)	String (6)	6
Surname	Member's surname	String (15)	15
Firstname	Member's firstname	String (15)	15
Category	Category of membership - S (senior) or J (Junior)	Char	2
Deleted	Has this member been deleted? Stores Y or N	Char	2
Record size	40 bytes		
Typical size of file	1 record for each member. With 400 members = 16,000 bytes		

Data validation
Data input by the user may not be entered correctly. Any check to ensure that it is acceptable is called data validation.

As I have no Visual Basic names for the controls yet, the identifiers below refer to those shown in the form design sketches earlier.

Control	***Form***	***Validation Check***
Membership No, (Adding a member)	Members	• Length 6 characters • Not already used for another member. This means that the MemberID field in the Members.dat file is a primary key field (i.e. a field that cannot contain duplicate values)
Membership No. (Deleting a member)	Members	• Must not be blank
Surname	Members	• Must not be blank
First name	Members	• Must not be blank
Membership Category	Members (list box)	• Must be Senior or Junior
Senior (rate per game)	Utilities	• Must be a number
Junior (rate per game)	Utilities	• Must be a number
Membership No. (start a game)	Game	• Must not be blank • Membership number must exist (on Members file). This is a file lookup check, and since the membership number will be stored on the Finished Games file, it is also an example of referential integrity.
Table No. (start a game)	Game (combo box)	• Must be the number of a table available for play. • A value must be selected from the combo box
Table No. (finish a game)	Game (combo box)	• Must be the number of a table currently in use • A value must be selected from the combo box

Entering the membership number of a new member lends itself to further validation. Since membership numbers consist of two characters followed by four digits I could check for this.
Visual Basic makes validating some data very easy by the use of list or combo boxes. In the three validation checks listed above that involve these controls, the user is forced to select a correct item of data. However I could still check that they have made a selection in the first place.

Maintenance: Modular Structure
Event procedures

Event	*Control*	*Processing*
Main form - buttons		
Click	Close	Closes program
Click	Game	Displays/loads Game form
Click	Membership	Displays/loads Members form
Click	Utilities	Displays/loads Utilities form
Click	Table Usage Report	• Prints report on table usage • Deletes Finished Games file (if user requests this)
Main form - other		
Load	n/a	Colour tables red or green according to whether or not they are in use, and displays starting time of game next to red ones.
Game form - Starting a game		
Load		• Populates combo box (tables available for play)
Checked Changed	Start radio button	Makes input controls for finishing a game invisible. Makes those for starting a game visible Changes text of button to 'Start Game'
Leave	Membership No. text box	• Displays member's name and category of membership
SelectedIndex Changed	Combo box	• Converts starting time of game to hours/minutes only Displays this starting time
Click	Start Game button	• Stores record of game in Current Games file • Changes table's colour to red and displays starting time next to it on the Main form • Populates combo box (with available table numbers)

Game .form - Finishing a game		
Checked Changed	Finish radio button	Makes input controls for starting a game invisible. Makes those for finishing a game visible. Changes text of button to 'Finish Game' • Populates combo box (with occupied table numbers)
Selectedlndex Changed	Combo box	• Retrieves record of game from Current Games file • Retrieves player's details from Members file • Calculates playing time in hours and minutes • Calculates cost of game • Converts starting and finishing times of game to hours/minutes • Displays start and finish time and cost of completed game
Click	Finish Game button	• Retrieves details of game from Current Games file • Updates record of finished table in Current Games file (sets • Occupied field to 'N') • Changes colour of table to green and removes starting time • Stores record of finished game in Finished Games file • Populates combo box (with occupied table numbers)
Members-form - Adding a member		
Checked Changed	Add radio button	Makes input control for deleting a member invisible. Makes controls for adding a member visible Changes text of button to 'Add Member'
Click	Add Member button	• Checks that membership number has not been used before • Stores new member's details in Members file

Members form - Deleting a member		
Checked Changed	Delete radio button	Makes input controls for adding a member invisible. Makes control for deleting a member visible Changes text of button to 'Delete Member'
Click	Delete Member button	• Deletes member from Members file
Members form - other buttons		
Click	Display Members	Displays details of all current members in list box
Click	Print Membership List	Prints report on current membership
Utilities form		
Checked Changed	Change Price of Game radio button	Displays controls for entering new senior and junior rates
Click	OK button	• Backs up Current Games and Finished Games files • Backs up Members file • Creates Current Games file • Stores new rates in Costs file

Game form's Load event, Finish radio button CheckedChanged event and Start / Finish Game button's Click event

Because the controls for starting a game will be visible when the form loads, the Load event should list the free tables in the combo box. When the user clicks the radio button for finishing a game the occupied tables should be listed. Both these tasks are similar and can be done by the same general procedure. No parameters are needed and nothing is returned so it can be a Sub procedure. I will name it ListTables.

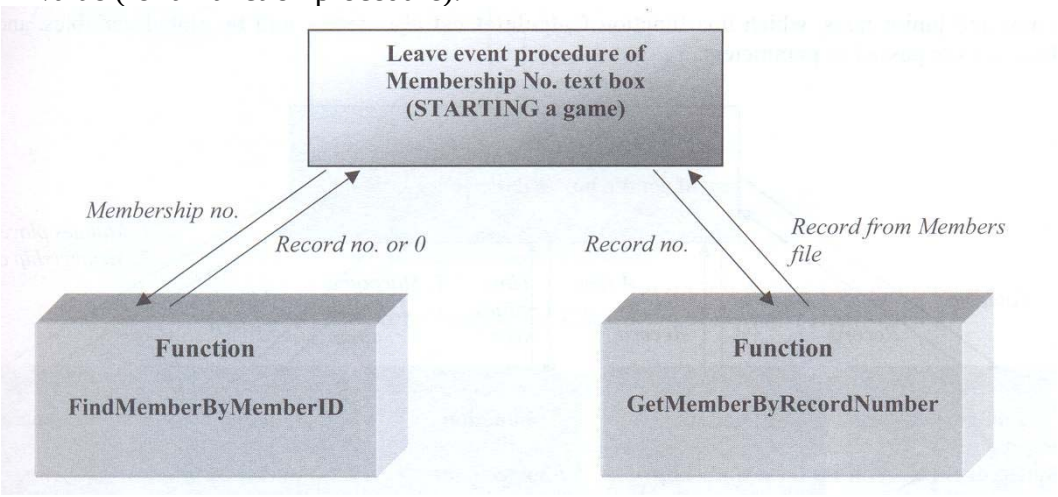
Name	ListTables
Type	Sub Procedure
Parameters / ReturnValue	None
Called from	Load event, Finish radio button CheckedChanged event and StartlFinish Game button's Click event on the Game form.
Purpose	If starting a game, populates the 'start' combo box with table numbers that are free. If finishing a game, populates the 'finish' combo box with table numbers that are occupied.

Leave event of Membership No. text box

This event is triggered when the user leaves the text box to select a table from the combo box. The Leave event of the membership number text box (for starting a game) should display the name and category of membership of the player whose membership number has just been entered. As it is possible that the membership number does not exist, I could consider breaking the task into two sub tasks, each in its own procedure:

- check to see if the membership number exists
- retrieve the record from the Members file

The second sub-task will only be done if the first sub-task reports that the number does exist. Since the 'parent' Leave event calls these procedures, the question is how the first sub-task reports back that the membership number exists or not. A function is an appropriate type of procedure and a Boolean return value looks promising. But if the number does exist, the next sub-task needs to know which record number in the file to go to. It could search the file for the membership number, but this has already been done by the first sub-task. A good solution is to make the return value of the first sub-task the record number in the file if the membership number is there, and another integer value if the number does not exist. A value of 0 will do (since record numbers begin at 1).
The diagram below uses a module structure chart to show the modular structure of this part of the program. An arrow going into a general procedure represents a value parameter and a return arrow represents a reference parameter (for a Sub procedure) or a return value (for a Function procedure).



Selecting from the combo box (starting a game) to display start time

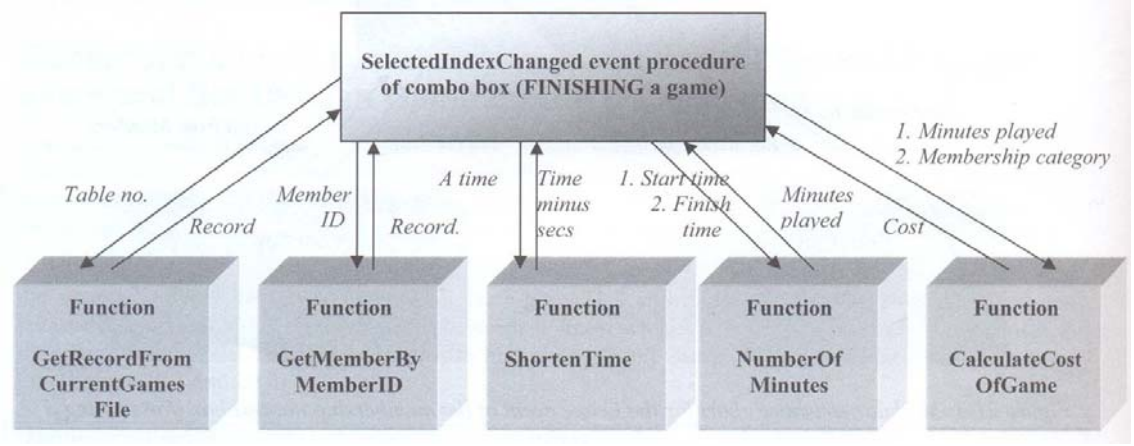
The start time of a game is taken from the system clock and this includes seconds as well as hours and minutes. The receptionist does not need the seconds part displayed so I will have a function, ShortenTime, that is sent a time and returns only the hours and minutes part. Note that other events also call it, and that an identifier for the parameter, FullTime, has been used to make the description of the purpose easier to write.

Name	ShortenTime
Type	Function
Parameters/ReturnValue	String value parameter (FullTime) - a time Returns String
Called from	Combo box SelectedIndexChanged event (both for starting and finishing a game) Main form's Load event
Purpose	Strips off the seconds Dart of FullTime and returns the hours and minutes Dart

Selecting from the combo box for finishing a game

Each of the five bulleted tasks for the SelectedIndexChanged event procedure can be put into a general procedure. The last one, to convert the starting and finishing times to hours and minutes only, has already been designed - Shorten Time above.

Each of the general procedures can be written as a function since they all return one item of data. The first function, to retrieve a particular table's details from the Current Games file, needs to be passed the table number. The second function, to retrieve a particular member's details from the Members file, needs to be passed a membership number. Function NumberOfMinutes needs to be passed the start and finish times of a game in order to calculate how many minutes the game took. Finally, to calculate the cost of a game requires the number of minutes it took (which has just been returned from NumberOfMinutes) and the category of membership.

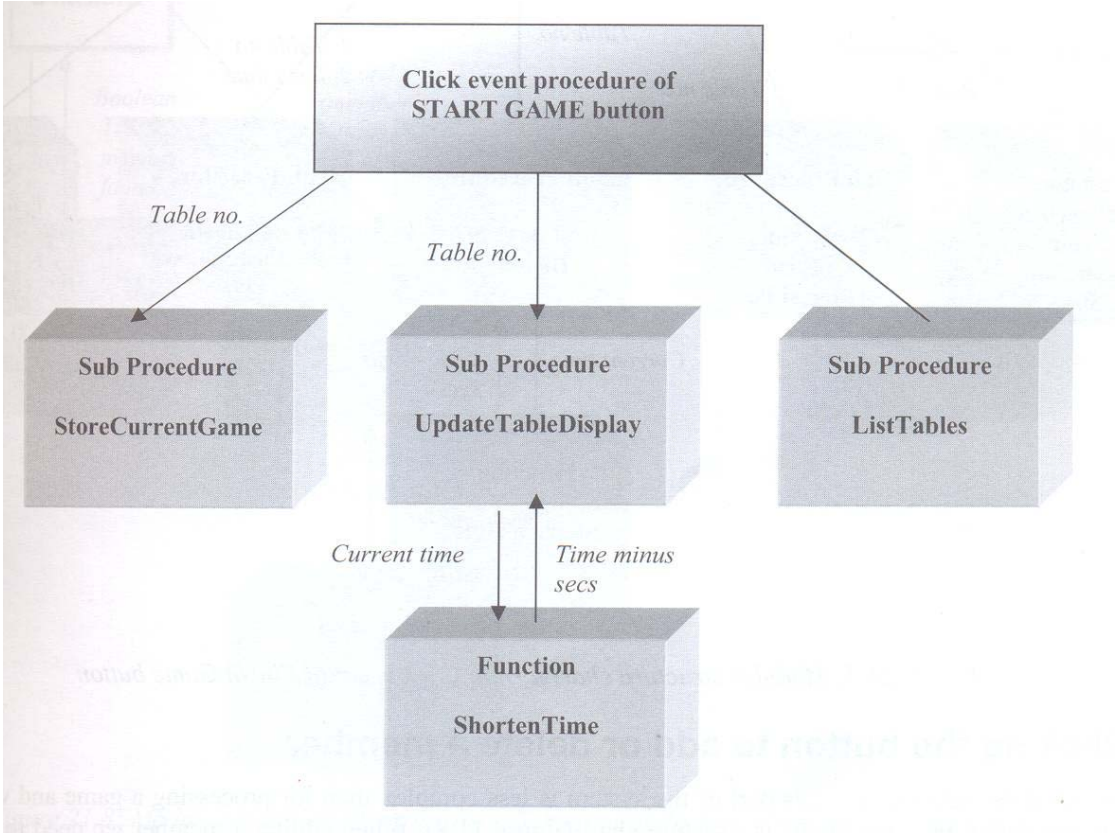


Clicking the Start Game button for a new game

There are three tasks that must be done when processing a new game:

- store details of the new game in the Current Games file
- change the table's colour to red and display the starting time repopulate the combo box so that the newly-used table is not listed

No data needs to be returned to the calling event procedure and so Sub procedures are used. ListTables does not need to be passed any parameters and so the line connecting it to its parent event procedure does not have an arrow. Note that UpdateTableDisplay in turn needs to call ShortenTime. This is so it can display the start time of the new game on the Main form.

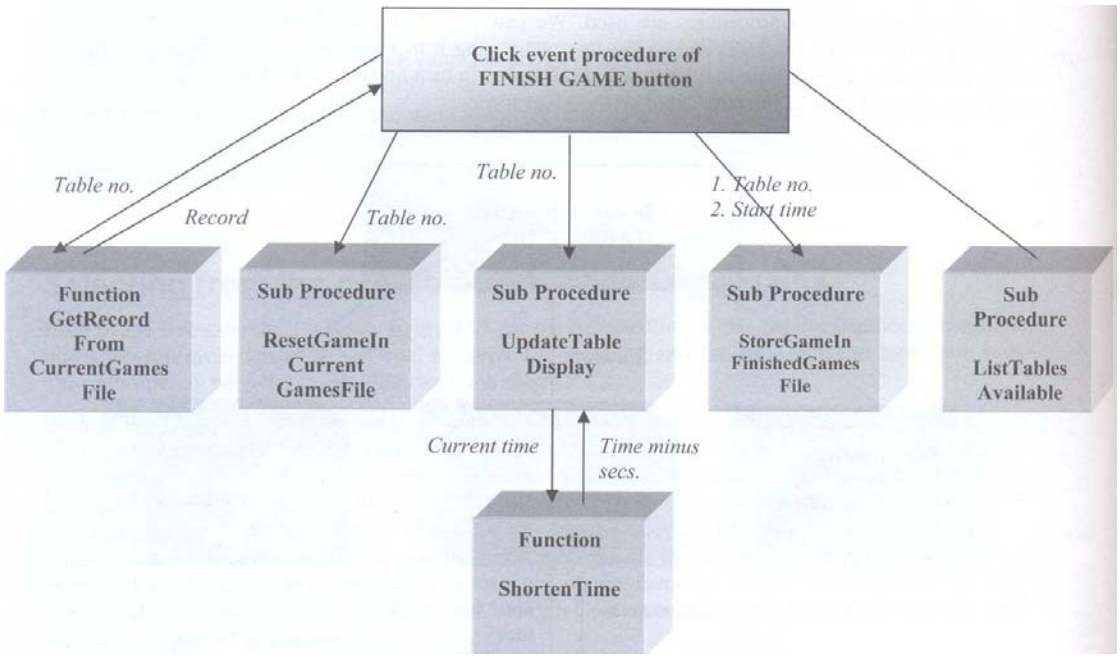


Clicking the Finish Game button for a finished game

Five tasks that must be done when the user clicks the button to process a finished game:

- retrieve details of the game that has finished from the Current Games file
- update the record of this table in the Current Games file so that its Occupied field stores 'N' .
- change the colour of the table to green and remove the starting time
- store details of the finished game in the Finished Games file
- repopulate the combo box with tables that are in use

I have already designed procedures to handle the first and third tasks. Since ResetGamelnCurrentGamesFile simply overwrites the occupied field with 'N', it only needs the table number. Procedure StoreGameInFinishedGamesFile needs the table number and the start time. The start time will have been retrieved in the record returned from GetRecordFromCurrentGamesFile.

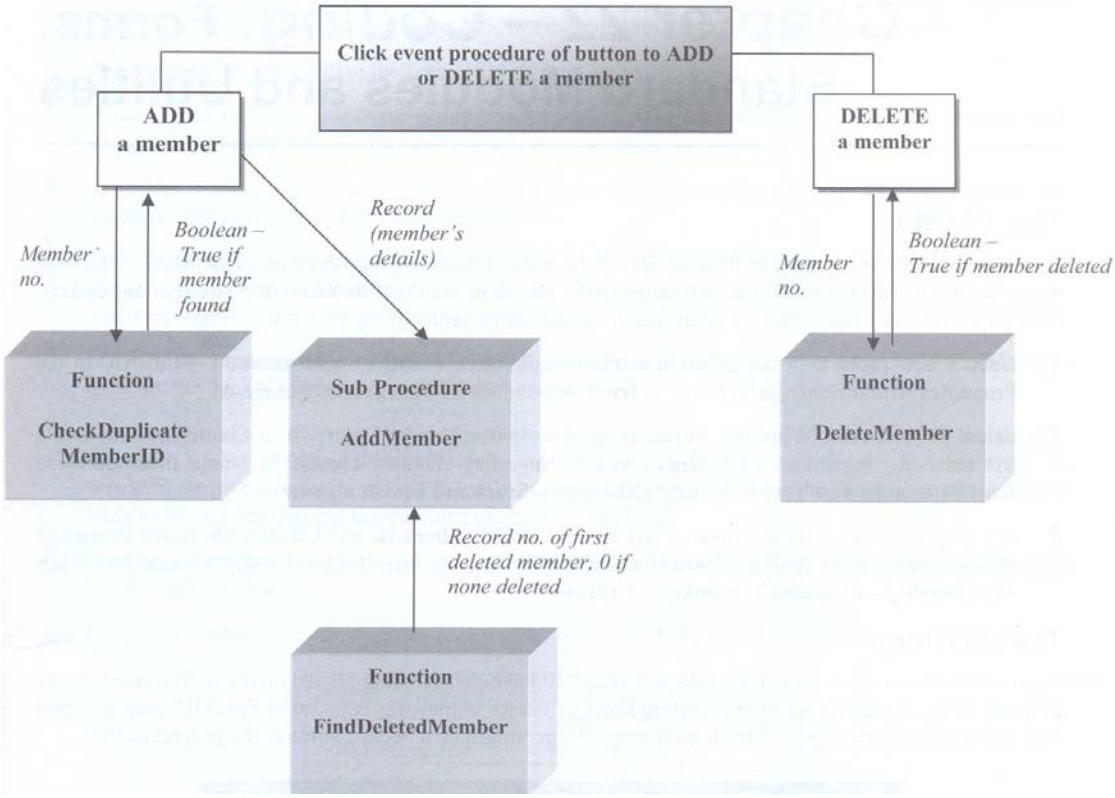


Clicking the button to add or delete a member

The modular structure for this part of the system is less complex than for processing a game and I can put everything into one modular structure chart. When adding a member I need to check that the new membership number does not already exist (unlikely but possible). This task is assigned to the function CheckDuplicateMemberID. It returns a Boolean value to indicate whether or not the membership number already exists.

Procedure AddMember actually writes the new record to the Members file, but recall that I decided to find the first logically deleted member (the Deleted field contains 'Y') and overwrite this record with the new one. The task of locating this deleted member is assigned to function FindDeletedMember. If there is a deleted one it returns the record number; if there isn't one it returns O. It is the job of AddMember to handle the 0 if this is returned.

Only one general procedure, DeleteMember, is used to process a deleted member. It is passed a membership number and searches the Members file for this number. If it finds the number it logically deletes the member and returns True. If it doesn't find the number the return value is False, and no changes are made to the file. Note that I could have had a separate function to check if the membership number exists in the file and given DeleteMember the single task of deleting the appropriate record. Instead I have packed two closely related tasks into DeleteMember.


Click event procedure of button to ADD or DELETE a member
ADD a member
DELETE a member
Member no.
Boolean – True if member found
Record (member's details)
Member no.
Boolean – True if member deleted
Function CheckDuplicate MemberID
Sub Procedure AddMember
Function DeleteMember
Record no. of first deleted member, 0 if none deleted
Function FindDeletedMember