SDD Major Project – Theory Scaffold/Examples

This guide is to be used with the Course Specifications booklet given to you, and the Sam Davis textbook.

Design Brief – This is a description of what your program will do (it needs to be at least 1-2 paragraphs long). It is in a user perspective rather than written for the developer (e.g. it doesn't need technical programming details).

Sample example:

The objective of this project is to produce a memory game that will be entertaining and to some extent capable of exercising a player's memory.
This will be carried out in the form of a common memory game in which players must remember the locations of a variety of dealt cards and then select pairs that contain the same images.

This should also include a means of objectively calculating the player's skill level in a scoring system of some sort. The program should also contain a variety of difficulty settings to cater for a wider user base.
It should be clear and intuitive to use through simple commands and easy to follow instructions throughout the game.

Design Specifications

Sketches

You need to include sketches for EVERY screen in your program.

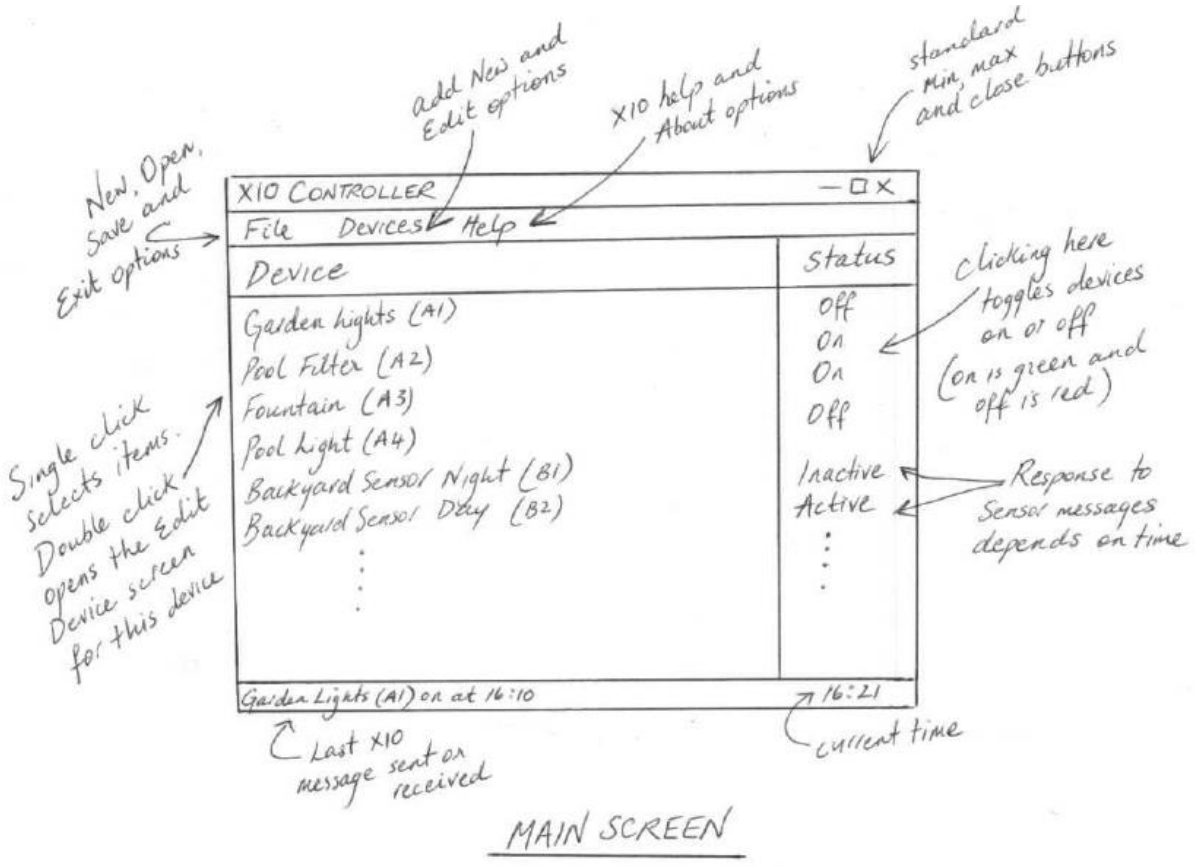
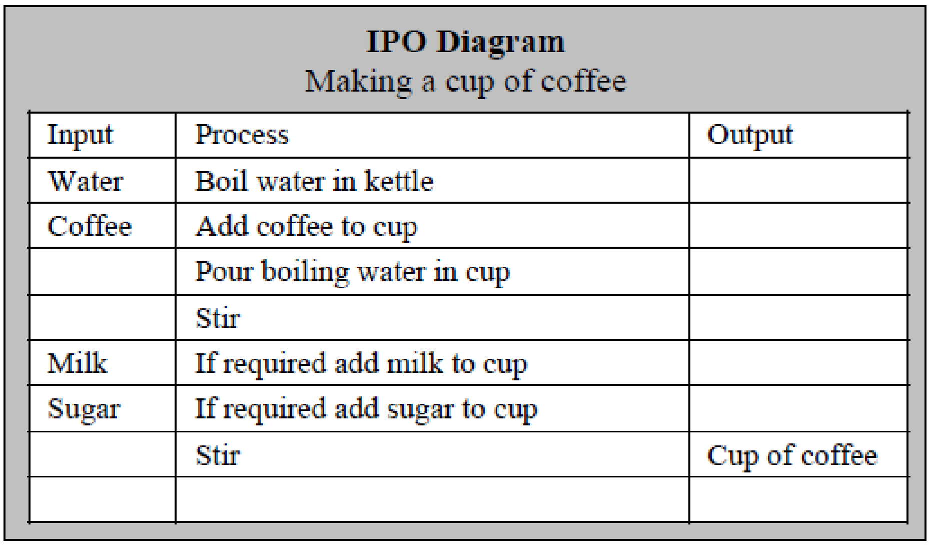

add New and Edit options
X10 help and About options
standard min, max and close buttons
New, Open, Save and Exit options
X10 CONTROLLER
File Devices Help
Device
Status
Garden lights (A1)
Pool Filter (A2)
Fountain (A3)
Pool Light (A4)
Backyard Sensor Night (B1)
Backyard Sensor Day (B2)
Off
On
On
Off
Inactive
Active
Clicking here toggles devices on or off (On is green and off is red)
Response to Sensor messages depends on time
Single click selects items. Double click opens the Edit Device screen for this device
Garden Lights (A1) on at 16:10
16:21
Last X10 message sent or received
current time
MAIN SCREEN

Fig 8.10
Initial design for the main screen.

IPO Diagram (Input Process Output)

For your program the inputs will be exact variables e.g. lblClear or ArrayNames. The process is what occurs and the output is what happens after the processing has been done (this could be another process or an out of a variable).



IPO Diagram
Making a cup of coffee

Input	Process	Output
Water	Boil water in kettle	
Coffee	Add coffee to cup	
	Pour boiling water in cup	
	Stir	
Milk	If required add milk to cup	
Sugar	If required add sugar to cup	
	Stir	Cup of coffee

Fig 4.3
IPO Diagram describing the inputs, processing and outputs required to make a cup of coffee.

System Flowchart

A system flowchart is NOT the same as an algorithm flowchart!

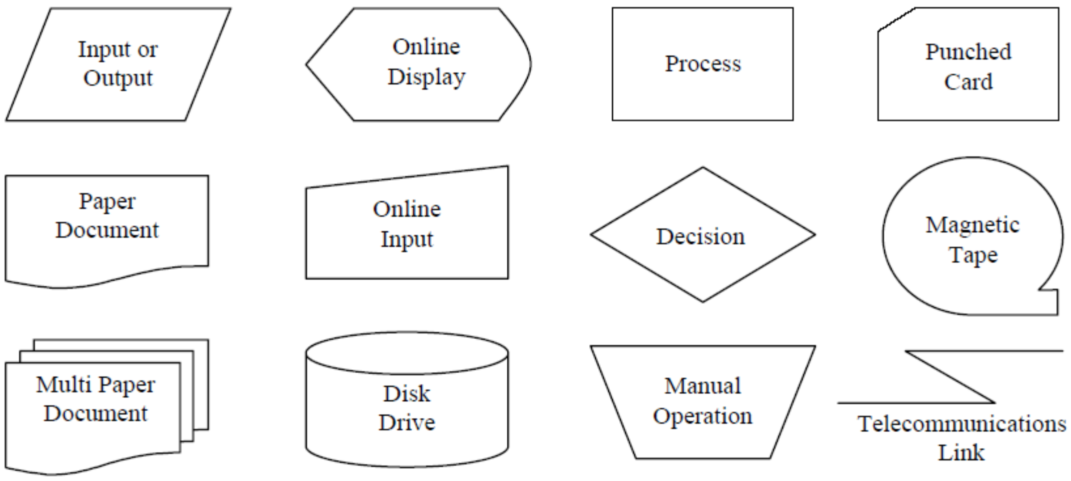


Fig 4.10
Components of System Flowcharts.

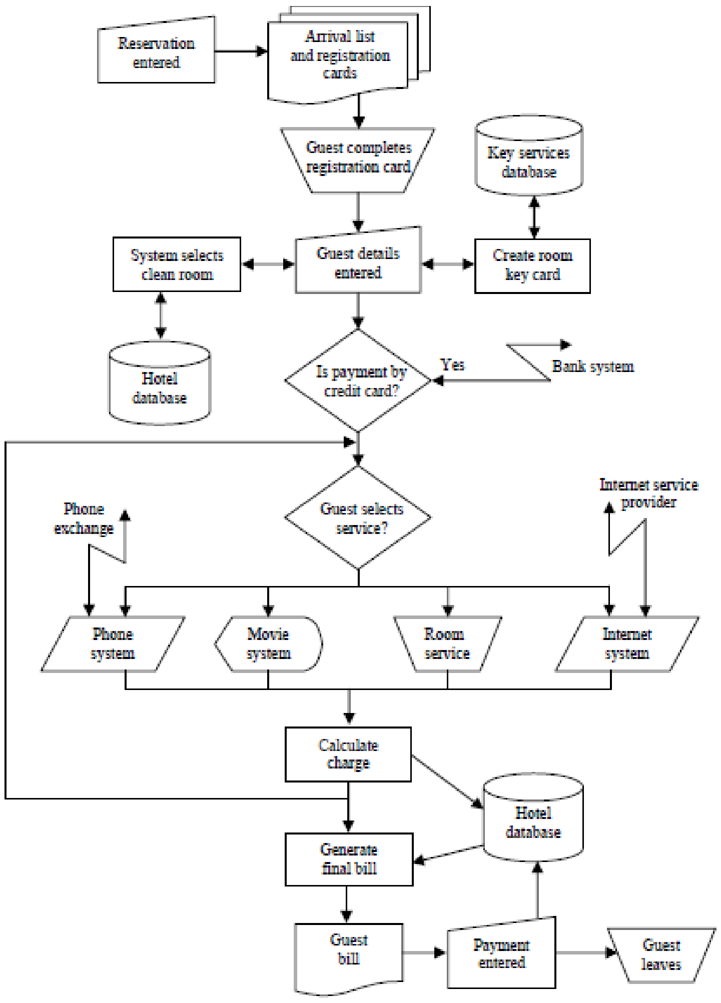


Fig 4.12
System flowchart describing the flow of data for an individual guest in a hotel.

Context Diagram

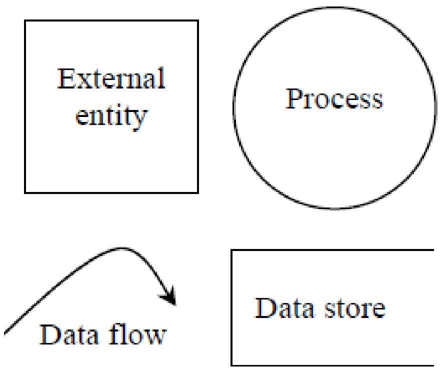


Fig 4.13
Symbols used on data flow diagrams.

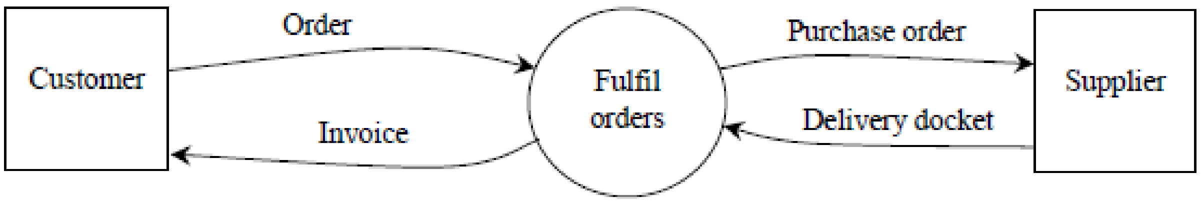


Fig 4.16
Context diagram for the book reseller problem.

Data Flow Diagram

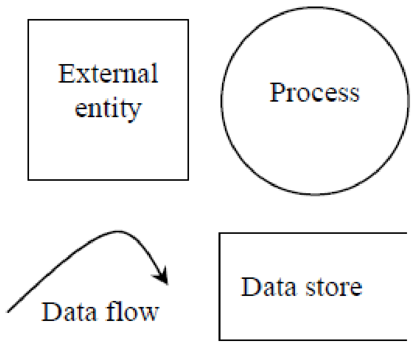


Fig 4.13
Symbols used on data flow diagrams.

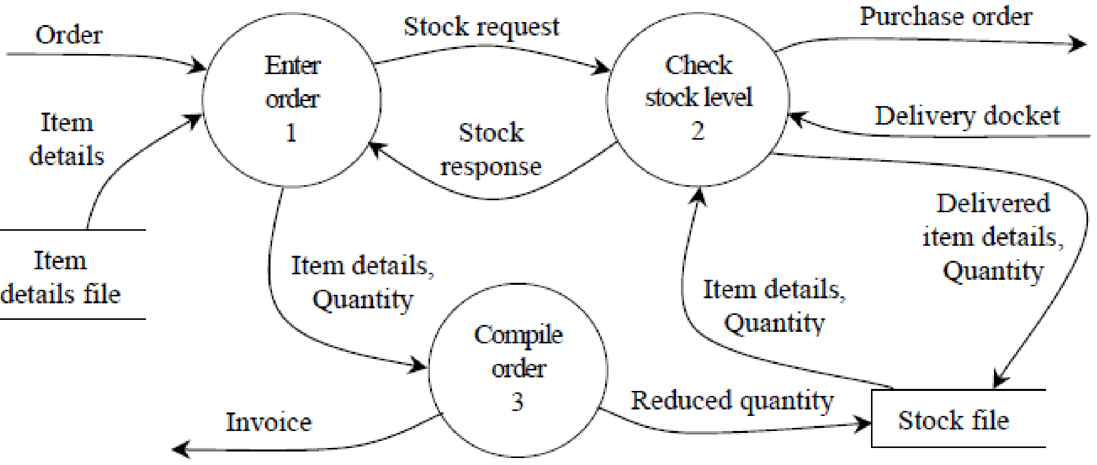


Fig 4.17
Level 1 data flow diagram for the book reseller problem.

Structure Charts

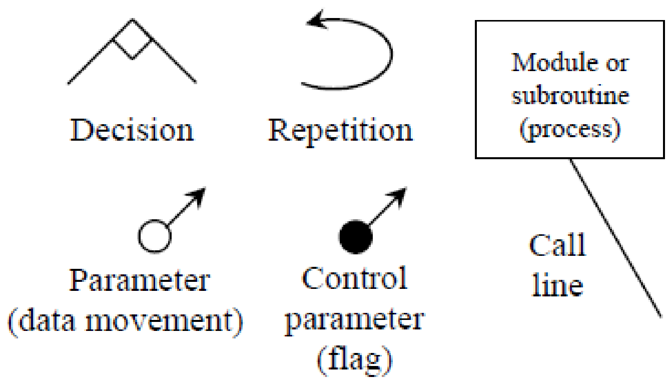


Fig 4.18
Symbols used in structure charts.

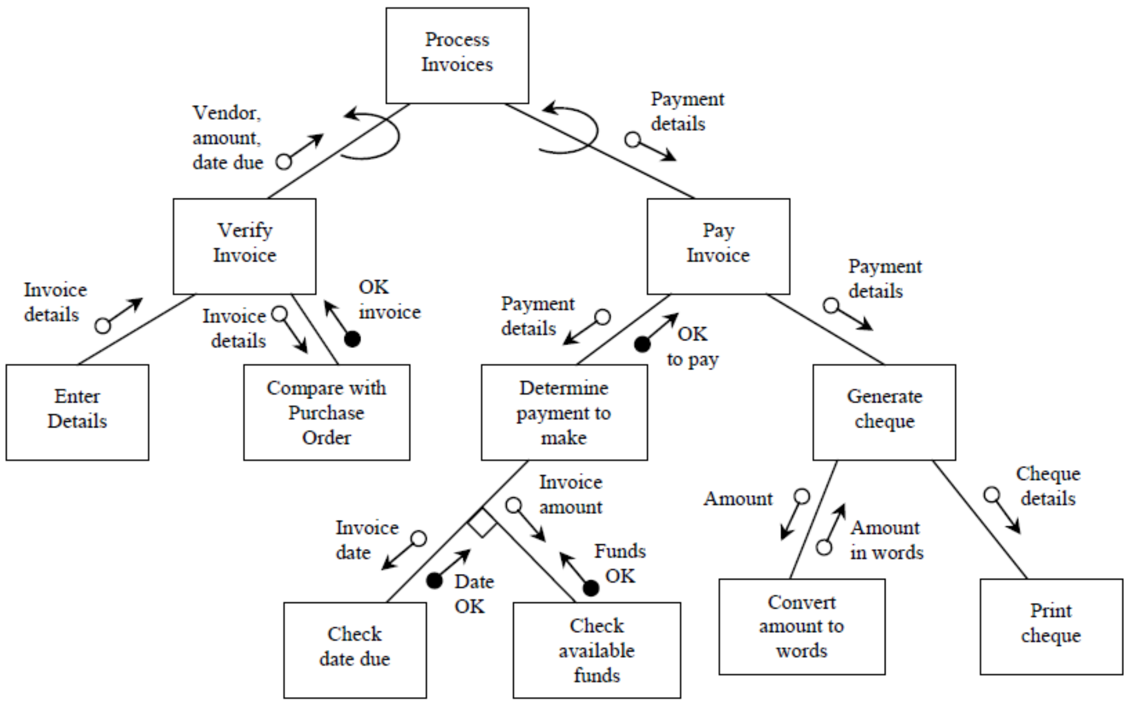


Fig 4.19
Structure chart for processing invoices.

Algorithm Design

You may write your algorithms as pseudocode OR flowcharts OR a mixture of both! You will need many separate algorithms for each module or procedure of your program.

Examples:

```
BEGIN MAINPROGRAM
  Get First
  Get Second
  Result = Biggest (First, Second)
  Display Result
END MAINPROGRAM

BEGIN Biggest(Item1,Item2)
  IF Item1 > Item2 THEN
    Big = Item1
  ELSE
     Big = Item2
  ENDIF
  RETURN Big
END Biggest
```

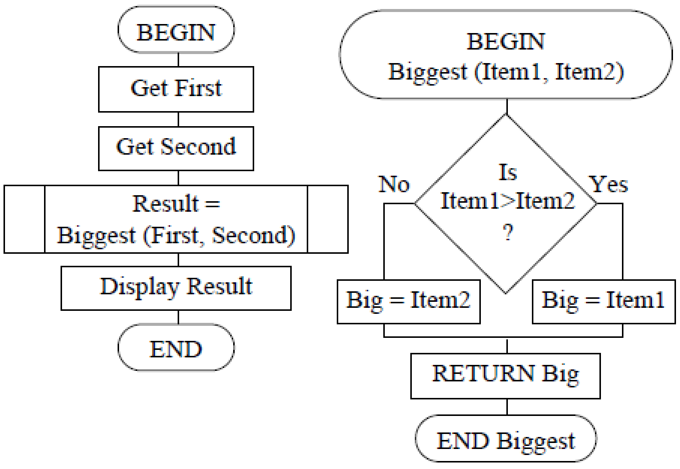


Fig 4.63
Using RETURN to pass values back in pseudocode and flowcharts.

```
BEGIN LoadArray
  Set Index to 0
  Get DataItem
  WHILE DataItem is not the sentinel
    Store DataItem in Item(Index)
    Increment Index
    Get DataItem
  ENDWHILE
END LoadArray
```

Fig 4.65
Algorithm to load data into an array.

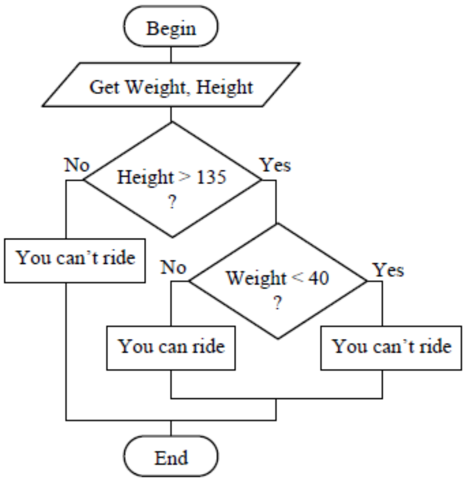


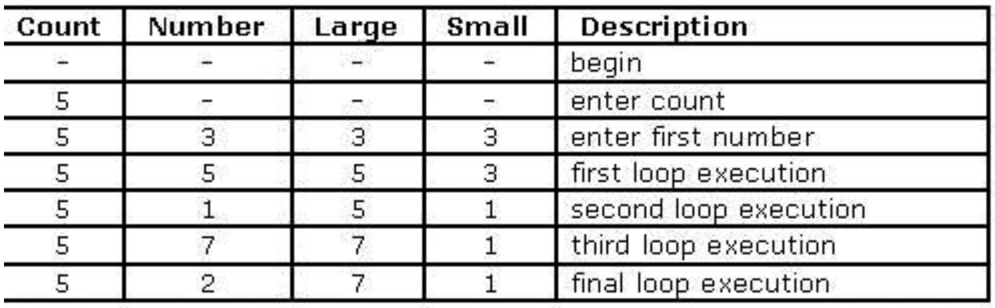
Fig 4.77
Algorithm to determine if a person can ride the roller coaster.

```
BEGIN MAINPROGRAM
 Input Number
 Sum = 0
 Tally = 0
 WHILE Number ≥ 0
   Sum = Sum + Number
   Tally = Tally + 1
   Input Number
 ENDWHILE
 Average = Sum/Tally
 Print Average
ENDMAINPROGRAM
```

Fig 4.75
Algorithm to calculate the average of a set of numbers.

Program Testing

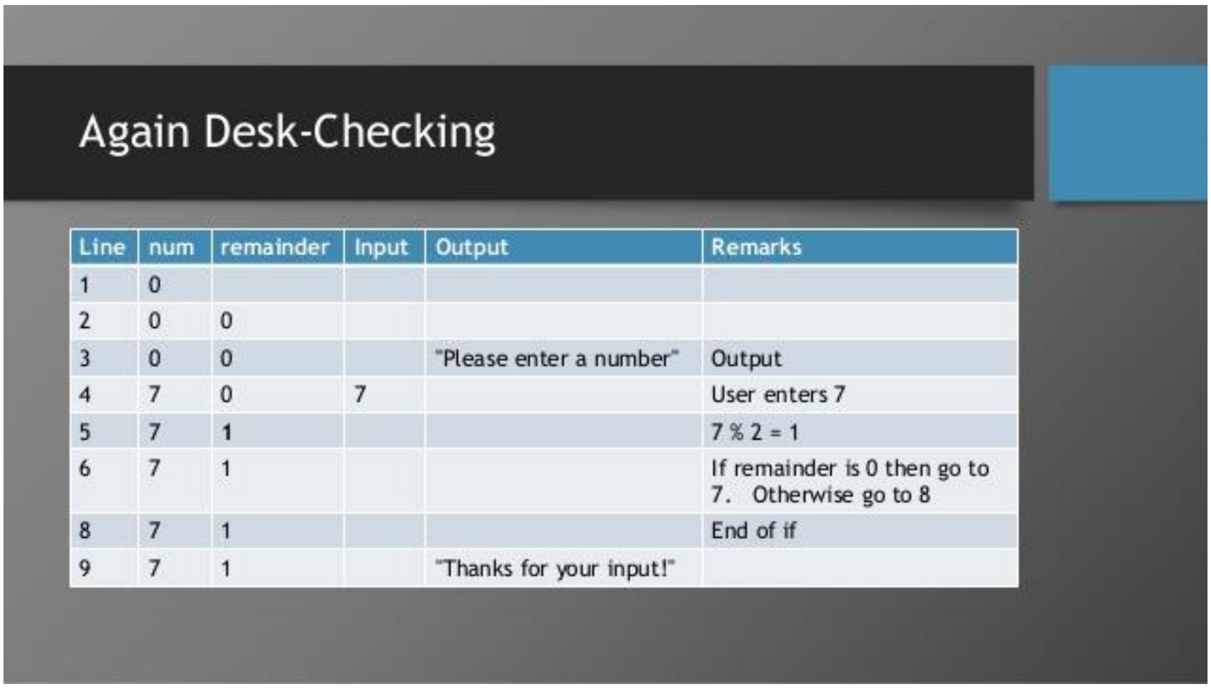
You need to design test data (sets of data to test below boundary, on boundary, above boundary, unexpected results etc.). You will use this test data to perform a desk check (you must check every possible thing a user might do!).



Count	Number	Large	Small	Description
-	-	-	-	begin
5	-	-	-	enter count
5	3	3	3	enter first number
5	5	5	3	first loop execution
5	1	5	1	second loop execution
5	7	7	1	third loop execution
5	2	7	1	final loop execution

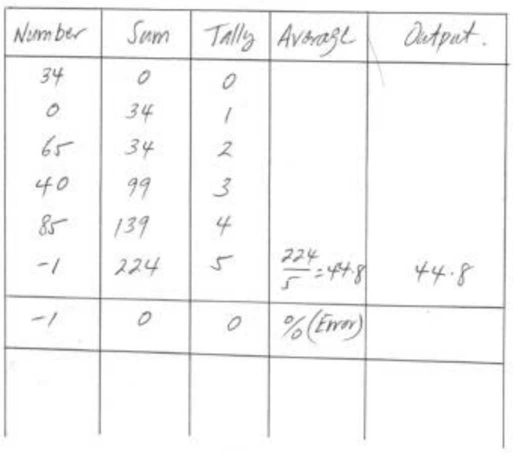
There are a few headings for a desk check. Usually it will include "input" on the left-hand side, or as above the input variables are count, number, large, small and description. Description is expected output.

In any desk check you need to have a heading for the inputs, an expected output (like description above) and an actual output column. Actual output will list what occurs (or what variables are returned) as a result of a pass or process occurring. If nothing happens during a stage, it is left blank or has a dash marking (-).



Again Desk-Checking

Line	num	remainder	Input	Output	Remarks
1	0				
2	0	0			
3	0	0		"Please enter a number"	Output
4	7	0	7		User enters 7
5	7	1			7 % 2 = 1
6	7	1			If remainder is 0 then go to 7. Otherwise go to 8
8	7	1			End of if
9	7	1		"Thanks for your input!"	



Number	Sum	Tally	Average	Output.
34	0	0		
0	34	1		
65	34	2		
40	99	3		
85	139	4		
-1	224	5	$\frac{224}{5}=44.8$	44.8
-1	0	0	% (Error)	

Fig 4.76
Desk check for the average algorithm.

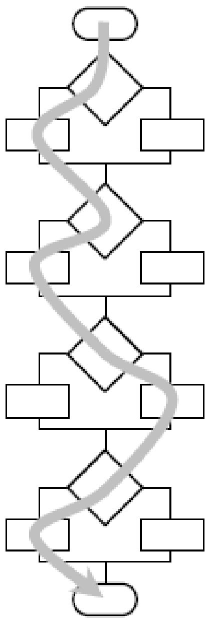


Fig 4.74
One of sixteen unique paths through this algorithm.

Index	First	Name(0)	Name(1)	Name(2)	Name(3)	Name(4)
		Fred	Mary	John	Amy	Ann
0	Fred	Mary				
1			John			
2				Amy		
3					Ann	
4						
						Fred

Program Testing Example

Navigation Buttons Testing Input, Expected Output, Actual Output table

Input	Expected Output	Actual Output
Index: Exit button	Close Program	Close Program
Index: Start button	Open OptionSelect Form	Open OptionSelect Form
OptionSelect: Learn The Rules button	Open RulesForm	Open RulesForm
Option Select: Learn How To Play button	Open HowtoPlayForm	Open HowtoPlayForm
Option Select: Clear Leader Boards button	Leader-board has only one high score after the next play through	Leader-board has only one high score after the next play through
Option Select: Play the Game button	Open GameForm	Open GameForm
RulesForm: Back button	Open RulesForm2	Open RulesForm2
RulesForm: Next button	Open OptionSelect Form	Open OptionSelect Form
RulesForm2: Back button	Open RulesForm	Open RulesForm
RulesForm2: Next button	Open RulesForm3	Open RulesForm3
RulesForm3: Back button	Open RulesForm2	Open RulesForm2
RulesForm3: Learn How to Play button	Open HowtoPlayForm	Open HowtoPlayForm
RulesForm3: Play the Game button	Open GameForm	Open GameForm
HowtoPlayForm: Back button	Open OptionSelect form	Open OptionSelect form
HowtoPlayForm: Next button	Open HowtoPlayForm2	Open HowtoPlayForm2
HowtoPlayForm2: Back button	Open HowtoPlayForm	Open HowtoPlayForm
HowtoPlayForm2: Next button	Open HowtoPlayForm3	Open HowtoPlayForm3
HowtoPlayForm3: Back button	Open HowtoPlayForm2	Open HowtoPlayForm2
HowtoPlayForm3: Next button	Open HowtoPlayForm4	Open HowtoPlayForm4
HowtoPlayForm4: Back button	Open HowtoPlayForm3	Open HowtoPlayForm3
HowtoPlayForm4: Next button	Open HowtoPlayForm5	Open HowtoPlayForm5
HowtoPlayForm5: Back button	Open HowtoPlayForm4	Open HowtoPlayForm4
HowtoPlayForm5: Next button	Open HowtoPlayForm6	Open HowtoPlayForm6
HowtoPlayForm6: Back to Option Select button	Open OptionSelect Form	Open OptionSelect Form
HowtoPlayForm6: Play The Game button	Open GameForm	Open GameForm
HowtoPlayForm6: Back button	Open HowtoPlayForm5	Open HowtoPlayForm5
GameForm: Back button	Open OptionSelect Form	Open OptionSelect Form
GameForm: Reset button	Open and Close GameForm	Open and Close GameForm
GameForm: Choose Difficulty combo-box choose Easy	Current instructions disappear, next instructions appear and Deal Cards button is enabled	Current instructions disappear, next instructions appear and Deal Cards button is enabled

	button is enabled	button is enabled
GameForm: Choose Difficulty	Current instructions disappear, next	Current instructions disappear, next

combo-box choose Hard	instructions appear and Deal Cards button is enabled	instructions appear and Deal Cards button is enabled
GameForm: Choose Difficulty combo-box type in "Random"	Nothing will happen and Deal cards button will not be enabled	Nothing will happen and Deal cards button will not be enabled
GameForm: Deal Cards button (Easy difficulty)	10 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up	10 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up
GameForm: Deal Cards button (Medium difficulty)	16 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up	16 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up
GameForm: Deal Cards button (Hard difficulty)	20 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up	20 Cards are dealt out and revealed at the end, then the 0 under the Time label starts counting up
GameForm: Start Counter button	All cards are hidden, the 0 under the Timer label stops counting and the game instructions disappear	All cards are hidden, the 0 under the Timer label stops counting and the game instructions disappear

Card Comparison Module Testing

This testing was undertaken solely to see what happens when each pair of card is selected. This is the results table when the test was undertaken

Yes means the cards were left unturned, points were deducted and the mistakes count increased
No means the cards were reverted back to normal, the cards

	Ant	Heart	Hourglass	Triceratops	Banana	Rocket	Planet	Monkey	Black wing symbol	OrangeBall
Ant	Yes	No	No	No	No	No	No	No	No	No
Heart	No	Yes	No	No	No	No	No	No	No	No
Hourglass	No	No	Yes	No	No	No	No	No	No	No
Triceratops	No	No	No	Yes	No	No	No	No	No	No
Banana	No	No	No	No	Yes	No	No	No	No	No
Rocket	No	No	No	No	No	Yes	No	No	No	No
Planet	No	No	No	No	No	No	Yes	No	No	No
Monkey	No	No	No	No	No	No	No	Yes	No	No
Black Wing Symbol	No	No	No	No	No	No	No	No	Yes	No
OrangeBall	No	No	No	No	No	No	No	No	No	Yes

The results were exactly as predicted meaning that the Card Comparison Module works as expected

Scores Test

Input	Expected output	Actual Output
Score 1500	New HighScore	New Highscore
Score 0	New HighScore	New Highscore

Score -100	No HighScore and no crashes	No Highscore and no crashes
Score -11700	No Highscore and no crashes	No Highscore and no crashes
Score -1000000000	No Highscore and no crashes	No Highscore and no crashest
Score -10000000000	No Highscore and no crashed	Game Crashed

This test was carried out by altering the BasePoint and IncorrectCardsPenalty variables to produce these results and see how the ResultsForm would react.

Notably at an extremely low score of -10000000000, the game crashed unexpectedly due to an arithmetic overflow, however, this was a hypothetical case that is very unlikely to happen. Because the program works properly with scores as low as -1000000000, this issue can be overlooked on the basis that this problem simply would not occur unless a user specifically aimed for such a low score.

Mathematically, since each mistake would reduce the score by 30, and it takes a minimum of 16*3 milliseconds to make a mistake, then it would take players 33333334 mistakes to achieve such a low time and at least 18.5 days of making mistakes non-stop for the user to make such a high number of mistakes.

Therefore it is reasonable to overlook this error.

<u>Insert Player Name tests</u>

These tests were carried out by entering various possible names in the insert Player name textbox in the InsertPlayerNames Form.

Input	Expected Output	Actual Output
	Messagebox "Please Insert Player Name"	Messagebox "Please Insert Player Name"
Name1	ResultsForm loaded with score in correct position and correct name	ResultsForm loaded with score in correct position and correct name
AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA	Messagebox "Please Insert a Shorter Name"	Messagebox "Please Insert a Shorter Name"
AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA AAAAAAAAAAAAA Repeated around 40000 times	Messagebox "Please Insert a Shorter Name"	Messagebox "Please Insert a Shorter Name" However the letters also disappeared
诶i 艾弗i fú 艾尺 (random Chinese characters)	The Name would be stored as either blanks or squares representing unknown	ResultsForm loaded with score in correct position and correct name

	characters	
मेस्सचुसेटस (random Indian characters)	The Name would be stored as either blanks or squares representing unknown characters	ResultsForm loaded with score in correct position and correct name
ろんど゛やりきはお (random Japanese characters)	The Name would be stored as either blanks or squares representing unknown characters	ResultsForm loaded with score in correct position and correct name

The Issue with the unexpected result of the letters disappearing upon reaching around 312000 characters is a minor issue that does not affect the functionality of the program at all.
The program still correctly displays the Messagebox “Please Insert a Shorter Name” and therefore this minor error can be overlooked.

The unexpected success with characters of different languages concluded the test and ultimately led to the conclusion that the InsertPlayerName Form modules work as expected

Coded Program

You will submit the entire solution folder, as well as a compiled (installable .exe) file. The compiling we will do together on the due date.

You need to also copy all parts of your code and paste into a Word document for submission.

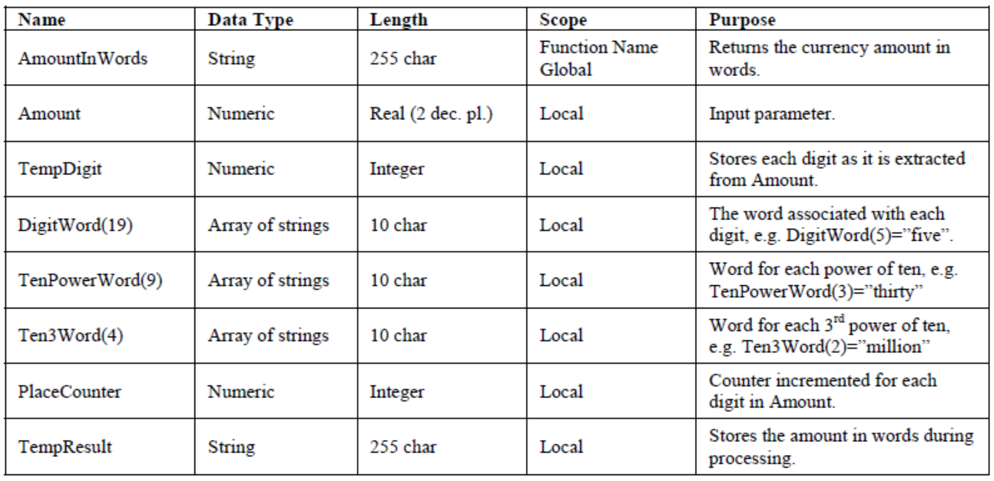
Documentation (refer to textbook pages 278-281)

You are to produce the following documentation:
A user manual which includes:

- An installation guide
- Instructions on how to use the program
- A troubleshooting guide (solving simple problems the user may encounter)

A technical manual that includes:

- A data dictionary for all variables used in your program



Name	Data Type	Length	Scope	Purpose
AmountInWords	String	255 char	Function Name Global	Returns the currency amount in words.
Amount	Numeric	Real (2 dec. pl.)	Local	Input parameter.
TempDigit	Numeric	Integer	Local	Stores each digit as it is extracted from Amount.
DigitWord(19)	Array of strings	10 char	Local	The word associated with each digit, e.g. DigitWord(5)="five".
TenPowerWord(9)	Array of strings	10 char	Local	Word for each power of ten, e.g. TenPowerWord(3)="thirty"
Ten3Word(4)	Array of strings	10 char	Local	Word for each 3rd power of ten, e.g. Ten3Word(2)="million"
PlaceCounter	Numeric	Integer	Local	Counter incremented for each digit in Amount.
TempResult	String	255 char	Local	Stores the amount in words during processing.

Fig 4.38
Data dictionary for the 'Convert Amount to Words' module.

- A listing of all procedure names with a description of what they are designed to do

Example of procedure names

ClearStoredInformation: Button activated module that returns all of the settings of Score, Mistakes, Time and Name back to default 0 values to essentially clear the leader-board.

ChooseDifficulty: Module triggered by changing the “Choose Difficulty” combo-box that reads which difficulty the user has selected and changes variables accordingly. For example if hard difficulty is chosen, this module changes the BasePoints variable to 155 and NumberofCards to 20.

MoveToPosition: Timer activated module that repeats every millisecond while activated to create a smooth animation of the cards approaching their designated locations. This is created by approaching the card one tenth of the way toward its designated location up until it reaches its location.

- A copy of all program code copy and pasted from Visual Basic (this is to safeguard against any potential issues)

Log Book (refer to textbook page 282)

You will be required to keep a log book that follows the format given to you by your teacher and depicts:

- All major milestones in the program’s development
- Records of any help that you receive, including links to website resources and tutorials
- Problems encountered:
  - Statement problem
  - Evaluation of possible solutions
  - Solution chosen and justification

The log book is to be completed throughout the program development. It will be sighted twice by your teacher before the final submission date.