UNIVERSITY OF SOUTHERN QUEENSLAND
CSC1401 - Foundation Programming (Semester 2, 2016)
Assignment II Specification

The *HomewareCity* Shopping Cart System

Due date: 9 Sep 2016

Weight: 12%

Type: Team-based (3 members)

Goals and Topics

The assignment problem is straightforward. All necessary details have been supplied. The solution of the problem will use the programming concepts and strategies covered in Workshops 1-6. The subgoals are:

- Obtaining advanced understanding of values, variables and arrays;
- Understanding program input and output, functions and expressions;
- Understanding simple strategies like iteration, validation, sum, count, minimum, and maximum plans;
- Translating simple design into JavaScript code
- The mechanics of editing, interpreting, building and running a program
- Testing a program
- Commenting source code
- Becoming confident and comfortable with programming in small problems

Background

In online marketing, a shopping cart is a piece of e-commerce software on a web server that allows visitors to an Internet site to select items for eventual purchase, analogous to the American English term "shopping cart." In British English, it is generally known as a shopping basket, almost exclusively shortened on websites to "basket".

The software allows online shopping customers to accumulate a list of items for purchase, described metaphorically as "placing items in the shopping cart" or "add to cart". Upon checkout, the software typically calculates a total for the order, including shipping and handling (i.e. postage and packing) charges and the associated taxes, as applicable.

The development of web shop systems took place directly after the Internet became a mass medium. This was a result of the launch of the browser Mosaic in 1993 and Netscape in 1994. It created

an environment in which web shops were possible. The Internet therefore acted as the key infrastructure developments that contributed to the rapid diffusion of the e-commerce. E-commerce (as a subset of e-business) describes all computer-aided business transactions. In 1998 a total of 11 e-business models were observed, one of which was the e-shop business model for a B2C (Business-to-consumer) business - also called the "online shop". The two terms "online shop" and "electronic" or "e-shop" are used inter-changeably. The term "online shopping" was invented much earlier in 1984; for example TV shopping often used the term before the popularity of the online method. Today the term primarily refers to the B2C transactional business model. In order to enable "online shopping" a software system is needed. Since "online shopping", in the context of the B2C business model, became broadly available to the end consumer, internet-based "online shops" evolved.

Although the most simple shopping carts strictly allow for an item to be added to a basket to start a checkout process (e.g. the free PayPal shopping cart), most shopping cart software provides additional features that an Internet merchant uses to fully manage an online store. Data (products, categories, discounts, orders, customers, etc.) is normally stored in a database and accessed in real time by the software.

Shopping Cart Software is also known as e-commerce software, e-store software, online store software or storefront software and online shop.

[*Source from: http://en.wikipedia.org/wiki/Shopping_cart_software*].

Introduction

HomewareCity is a fast growing family business in Toowoomba. In the past years, *HomewareCity* has served Toowoomba local community well. Aiming to promote customers' shopping experience and dealing with increasing demands raised by busy, aged, or disability customers, *HomewareCity* is going to introduce online shopping facility to the community and thus, need to develop a shopping cart system to allow customers to shop online. Against a group of talented programmers, you really want to win the contract to develop the system. But firstly you need to develop a prototype program to demonstrate the main functionality of the system, as required by *HomewareCity*.

To assist development of the program a template written in HTML and JavaScript has been provided by *HomewareCity*. You can download it HERE. Note that *HomewareCity* has specifically required that the program is going to be developed in JavaScript and all code should go into the $< script > \cdots < /script >$ section.

Functional Requirements

As the prototype for a web-based shopping cart system, the program is to be implemented in $JavaScript$ and running on $Firefox$, an operating system independent web browser. *HomewareCity* has specified the following business requirements:

1. The program should be running without errors throughout two Phases: *Information Gathering* and *Information Presenting*.

2. In *Information Gathering*, each time the user adds one product item to the shopping cart. The program should first confirm with the user for willingness of shopping before proceeding to gather information of the product item in purchase.
3. When receiving an order, the program should first prompt and ask the user to enter the product code before adding the product item to the shopping cart. If the user enters an invalid value, for example, a non-number value or a non-existing product code, the program should alert an error message on screen and then prompt the user for re-enter. The process should iterate until a valid product code is entered.
4. If the entered product code is valid, the program should then prompt the user to input the quantity value in purchase. Again, if an invalid value is input, such as a non-number value, a negative number or zero, the program should display an error message then iterate until receive a valid number for quantity.
5. After valid input of product code and quantity, the program should add the product item into the shopping cart, and then loop back to seek user confirmation for either proceeding further to add one more item or moving to the *Information Presenting Phase* for check-out.
6. If the user confirms not to shop anymore, the *Information Gathering Phase* is completed and the program then moves to *Information Presenting*.
7. In the *Information Presenting Phase*, the program prints on the web page a table containing the product items in the shopping cart, including information such as product names, prices, quantity, and cost.
8. To make the Shopping Cart System user-friendly, *HomewareCity* also expects the program to display some statistic information:
    - The total amount for ordered items in the shopping cart;
    - The average cost per item in the cart;
    - The most expensive product item;
    - The least expensive product item.

Respectively, Figure 1 and 2 illustrate the data flow in *Information Gathering Phase* and a sample result presented to the html page in *Information Presenting Phase*.

Arrays

Two arrays have been supplied in the template provided by *HomewareCity*, one for the list of products (namely *productListArr*) and the other for their corresponding prices (namely *priceListArr*). Note that in the two arrays a product item and its corresponding price share exactly the same index in their respecting arrays[1].

You need to create two arrays – one to store the product code of ordered products (namely *orderedProductCodeArr*), the other to store the quantity of ordered products (namely *quantityArr*). Just like *productListArr* and *priceListArr*, an ordered products' code and quantity will be stored at exactly the same index in their corresponding arrays. A diagram illustrating the relationships between *productListArr*, *priceListArr*, *orderedProductCodeArr*, and *quantityArr* is shown in Figure 3.

[1]Hint: there is no need for an array to store the product codes. They are just the index in *productListArr*.

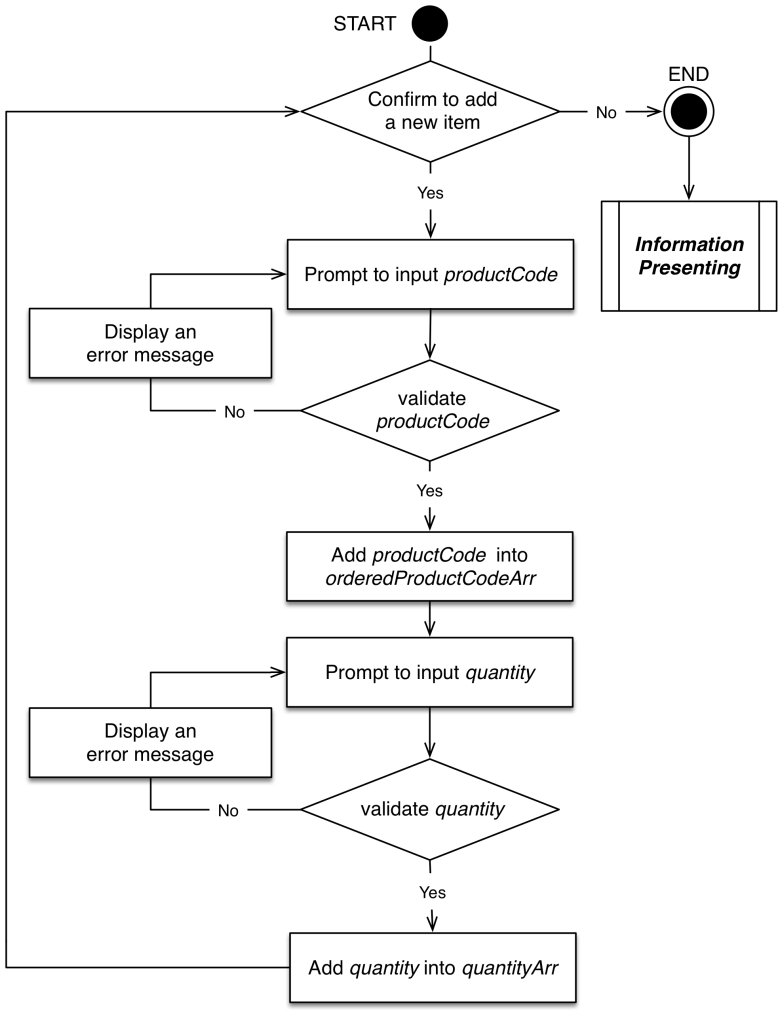

Figure 1: Data Flow in Information Gathering

Shopping Cart

PRODUCT	PRICE	QUANTITY	COST
Tea Set	39.95	3	119.85
Carioca Cups	54.95	1	54.95
Cake Pop Scoop	9.95	4	39.80
Steel Garlic Press	34.95	2	69.90
Multi Grater-Detachable	26.95	2	53.90
Pasta Machine	144.95	1	144.95

Summary

- The total cost amount is $483.35
- The average cost of purchased items is $37.18
- The most expensive purchased item is '*Pasta Machine*'
- The least expensive purchased item is '*Cake Pop Scoop*'

Figure 2: Sample Result of Information Presenting

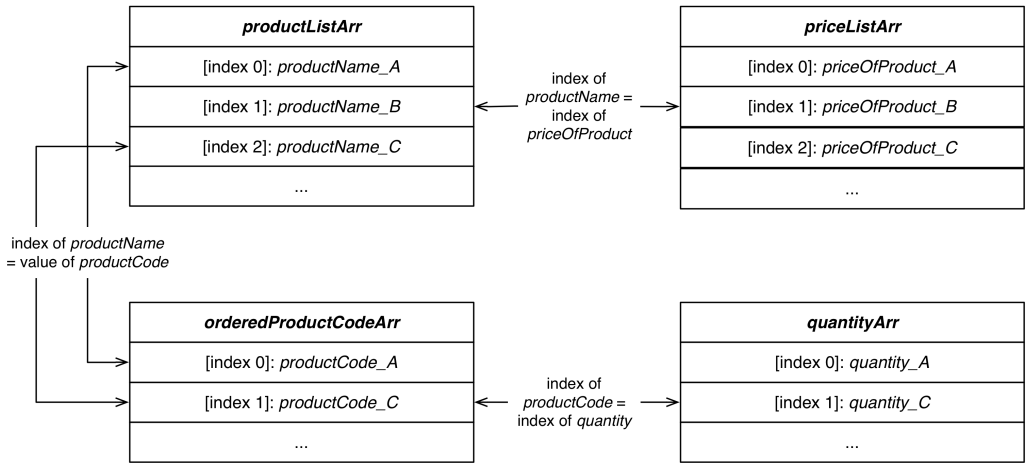

Figure 3: Relationship Diagram for Arrays

Implementation

Task 1 - Presenting the Catalogue

In the beginning of the program, print to a table the catalogue for all products including name, code, and price, to assist users shopping. You should use a loop plan to access the data stored in *productListArr* and *priceListArr* and present it in a four-column table, like Figure 4, where each column is a set of products with code, name, and price.

Catalogue

CODE	PRODUCT	PRICE	CODE	PRODUCT	PRICE	CODE	PRODUCT	PRICE	CODE	PRODUCT	PRICE
0	Salad Server Set	18.70	1	Party Serviette Holder	11.95	2	Tea Set	39.95	3	Mixing Bowl Set	49.95
4	Knife Block Set	99.95	5	Coffee Capsule Holder	29.95	6	Plastic Sensor Soap Pump	79.95	7	Storage Bucket	24.95
8	Oven Glove	9.95	9	Apron	29.95	10	Biscuit Barrel	39.95	11	Chopping Board	12.95
12	Carioca Cups	54.95	13	Soup Bowls	43.00	14	Elevate Wood Turner	19.95	15	Pasta Machine	144.95
16	Teapot	29.95	17	Cake Pop Scoop	9.95	18	Cookbook Stand	29.95	19	Chocolate Station	34.95
20	Coffee Maker	29.00	21	Pepper Mill	84.94	22	Salt Mill	84.95	23	Glass Storage Jar	4.95
24	Measuring jug	19.95	25	Kitchen Scale	39.95	26	Tenderiser	34.95	27	Pizza Docker	19.95
28	Knife Sharpener	79.95	29	Steel Cork Opener	36.95	30	Steel Garlic Press	34.95	31	Steel Can Opener	36.95
32	Stainless Steel Crank Flour Sifter	33.95	33	Mineral Stone Mortar and Pestle	74.95	34	Citrus Cather	19.95	35	Cherry & Olive Pitter	27.95
36	Multi Grater-Detachable	26.95	37	Stainless Steel Colander	44.95	38	Steel Pizza Pan	12.95	39	Pop Container	22.95

Figure 4: Sample Catalogue in Display

Task 2 - A Validation Plan for Valid Product Codes

You need to implement a validation plan to get a valid input from the user for product code. An input value is considered invalid if:

- it is not a number at all;
- it is not an integer number;
- it is a negative number;
- it is a number out of range (greater than or equal to the size of *productListArr*).

Task 3 - A Validation Plan for Valid Quantity Values

You need to implement another validation plan to get a valid input from the user for quantity. An input value is considered invalid if:

- it is not a number at all;
- it is not an integer number;
- it is zero or a negative number;
- it is greater than 100.[2]

[2]Orders with quantity of greater than 100 will need to go wholesale and will not be accepted here by the system.

Task 4 - An Iteration Plan for Information Gathering Phase

You need to design a looping plan to implement the *Information Gathering Phase*. Refer to Functional Requirements and Fig. 1 for the detail of data flow in the loop. Clearly, this task should incorporate the works in Task 2 and 3.

Task 5 - A Sum Plan to Calculate the Total Cost

Your program needs to calculate the total cost for all ordered products in the shopping cart. The calculation formula is provided:

$$totalCost(cart) = \sum_{i \in cart} price_i \times quantity_i$$

Task 6 - A Maximum Plan to Find the Most Expensive Product in Cart

Your program needs to find the most expensive product in the shopping cart. To do it, for each of the products added in to the cart you need to retrieve the corresponding price from *priceListArr*, and then compare the prices one by one. Once you find out the most expensive price, the corresponding product item in *productListArr* will be the most expensive product in the shopping cart.

If you have multiple products in the cart sharing the same most expensive price you can select any one of them as the "most expensive product".

Task 7 - A Minimum Plan to Find the Least Expensive Product in Cart

Your program also needs to find the least expensive product in the shopping cart. You may adopt the the same strategy in previous task for this task. Again, if you have multiple products in the cart sharing the same least expensive price you can select any one of them as the "least expensive product".

Task 8 - An Average Plan to Calculate the Average Cost

Your program needs to be able to calculate the average cost for all product items in the shopping cart. This can be done by the total cost divided by the accumulated value of quantities:

$$averageCost(cart) = \frac{\sum_{i \in cart} price_i \times quantity_i}{\sum_{i \in cart} quantity_i}$$

You should handle the "Division by Zero" exception when calculating average.

Task 9 - Presenting the Cart

Print to a table the detailed information of shopping cart, such as product name, price, quantity, and cost. You should adopt an iteration plan to visit the elements stored in *quantityArr* and *orderedProductCodeArr* in order to get the index to retrieve product names and prices from *productListArr* and *priceListArr*.

All currency values should be at cents in terms of precision.[3]

Task 10 - Presenting the Statistics

Print to an unordered list the statistic information (total cost, most expensive item, least expensive item, and average cost per unit for the ordered product items) on shopping cart.

The table and the list should be formatted like the screenshot in Figure 2. The same as that in Task 9, all currency values should be at cents in terms of precision.

Challenging Task - Duplicate Order Detection [Optional]

Note: No extra marks are gained for completion of the challenging task.

HomewareCity will appreciate it if an extra feature can be delivered to detect duplicate orders. If an ordered item has already been in the cart, the system should detect it, and then ask for user confirmation for updating the quantity or not. If the user gives a positive confirmation, the system will proceed to prompt for quantity and then replace the stored quantity by the newly entered value; otherwise, the program terminates the current product-adding process and loop back to ask user confirmation for adding a new item or not. Note that the user is not allowed to completely remove an ordered item from the shopping cart.

Program Integration Test

You need to test the program for all functionality thoroughly before delivering the program. The program should be running appropriately without any syntax and logic errors.

Non-Functional Requirements

- All code should appear in the script section in the head of the HTML document. Do not write any code in the HTML body. All functionality are delivered by JavaScript.
- In the script order your code as follows:
  (a) Constants;
  (b) Variables and objects (declared and initialised);
  (c) Other statements.

[3] Round up and keep only two digits after decimal point.

- Variable and constant identifiers should use an appropriate convention. Identifiers should be written consistently throughout the code.
- Code is indented appropriately and grouped in blocks according to the common tasks attempting to.
- Appropriate comments should be added to all blocks of code. Do not simply translate the syntax into English for comments, instead, describe the purpose of the code block.

Submission

What A Team Needs to Submit – Two Files

A team makes only one common submission, in which all members should make equal contributions and share the same mark. Assignment Submission Portal will open to only registered teams. In other works, students without a registered team will not be able to submit the assignment and receive any marks or feedback.

For a complete submission you need to submit two files as specified below. You can submit them individually or compress them into a *.zip* (or *.rar*) file and submit it. The assignment submission system will accept only the files with the extensions specified in this section.

1. ***Statement of Completeness*** in a file saved in *.doc, .docx, .odt, .rtf* or *.txt* format in 200-300 of your own words describes the following issues. You should first specify the registered name of the team and all members' name and student ID on the top of the statement.
    - **The state of your assignment**, such as, any known functionality that has not been implemented, etc. (It is expected that most teams will implement all of the functionality of this assignment.)
    - **Problems encountered**, such as, any problems that you encountered during the assignment work and how you dealt with them. This may include technical problems in programming and people-soft problems in team working;
    - **Reflection**, such as, any lessons learnt in doing the assignment and handling teamworking and suggestions to future programming projects.
2. ***The program*** in a file saved with an *.html* extension contains the source code implemented following the functional and non-functional requirements.

Late Submission and Extension Request

Please refer to *USQ Policy Library - Assessment Procedure* for information on the late submission policy and *USQ Policy Library - Assessment of Compassionate and Compelling Circumstances Procedure* for considerable special circumstances in extension request.

The Extension Request Form is available on the course's StudyDesk. Should you need to request an extension please fill the form and email it to the Course Examiner with supportive documents (e.g., medical certificate or endorsement letter from supervisor in workplace) prior to the due date . **Please note that any requests without supportive documents will be declined straightway without consideration**.

Marking Criteria

The assignment will be marked out of 24 and scaled down to a grade out of 12. Table 1 presents the marking criteria. If all criteria are satisfied you will receive 24 marks. If not all criteria are met, part marks may be given. Check your own submission against these criteria before you submit.

Table 1: Marking Criteria

ID	**REQUIREMENTS**	MARK
	Statement of Completeness	
1	The statement is in appropriate length of 200-300 of own words covering issues on both programming and team working	1
2	The "State of assignment" reflects the true state of completeness	1
3	The "Problems encountered" discusses problems and dealing strategies	1
4	The "Reflection" discusses learned lessons and reasonable suggestions	1
	Subtotal	*4*
	Functional Requirements	
5	The program is running without any errors	1
6	Task 1 - The Catalogue is presented appropriately using an iteration plan	1
7	Task 2 (a) - User input for product code is obtained as required and parsed to appropriate value type if necessary	1
8	Task 2 (b) - The validation plan validates product code inputs as required	1
9	Task 3 (a) - User input for quantity is obtained as required and parsed to appropriate value type if necessary	1
10	Task 3 (b) - The validation plan validates quantity inputs as required	1
11	Task 4 (a) - Information Gathering Phase in iteration takes multiple products	1
12	Task 4 (b) - User confirmation is obtained using correct function and used as the sentinel for iteration plan	1
13	Task 5 - The sum plan calculates the total cost correctly	1
14	Task 6 - The maximum plan finds the most expensive product in cart	1
15	Task 7 - The minimum plan find the least expensive product in cart	1
16	Task 8 (a) - The average plan calculate the average cost correctly	1
17	Task 8 (b) - "Division by Zero" exception is handled appropriately	1
18	Task 9 (a) - The cart is presented completely in table form as required	1
19	Task 10 (a) - The statistics are presented completely in unordered list as required	1
20	Task 9 (b), 10(b) - All currency values are displayed at cents in precision	1
	Subtotal	*16*
	Non-functional Requirements	
21	All functionality are implemented in JavaScript. No code goes outside of the script section	1
22	Identifiers of variables and constants are following professional conventions	1
23	Constants and variables are used in calculation and expression instead of explicit values	1
24	At least five comments are added to describe the purpose of blocks of code	1
	Subtotal	*4*
	TOTAL	**24**

Team Registration

Express to the "*Forum for finding members to form a team*" your interest in either recruiting people to form a team or joining a team with vacant positions. Once you have a team formed, please fill in the "*Team Registration Form*", which is available on StudyDesk, and email it to the Assistant Examiner. As a sign of registration completion, you should find your team name on the "*Registered Teams*" table under Assignment 2 on StudyDesk, as well private forum and chatroom opened for your team discussions.

Team-working Styles

Figure illustrates two styles in team-based programming working. You may choose either one of them or rely on any other styles as long as it suits your team. Please refer to the video recording of "*Workshop 1a - Important Stuff in CSC1401*" for oral description of these working styles and raise up any related issues to the assignment's public forum for discussions.

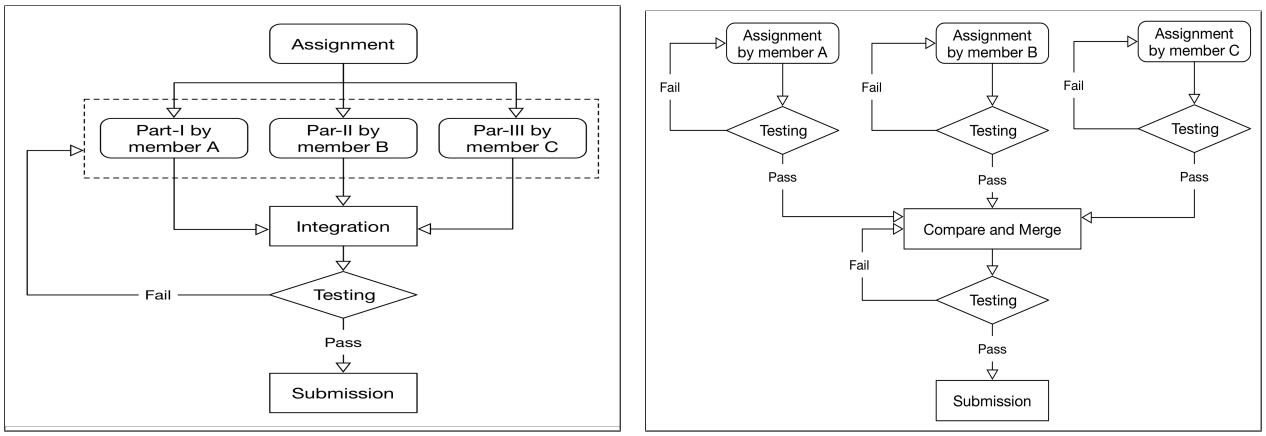

Figure 5: Team-working Styles

Team Conflict Resolution

It is observed in many projects that a team may experience conflict or even crisis during teamwork, because members are individuals with different backgrounds and personalities. While trying to resolve a conflict in your team, please follow the 2-Stages strategy:

Stage I - Internal Resolution Try to resolve the conflict within the team. Figure 6 illustrates the steps to conflict resolution – a variation of the model introduced by Joseph Phillips in 2010 [4] – that you should take while trying to resolve a conflict.

[4] Joseph Phillips (2010). IT Project Management: On Track from Start to Finish, Third Edition, Chapter 7 - and Chapter 8 - Managing Teams. McGraw-Hill Osborne Media.

Stage II - Report to Course Examiner You should upgrade the action to Stage II only after all means of effort in Stage I have been taken and failed. While it is necessary, please fill in the "*Crisis Report*", which is available on StudyDesk, and send it to the Course Examiner with supportive evidence such as meeting minutes and communication records. **Note that a report without supportive evidence will be declined because it may contain bias and puts somebody else in an unfair, disadvantaged position**. The Course Examiner will then be engaged and based on investigation, make a final decision.

Please refer to the video recording of "*Workshop 1a - Important Stuff in CSC1401*" for oral description of the conflict resolution strategy.

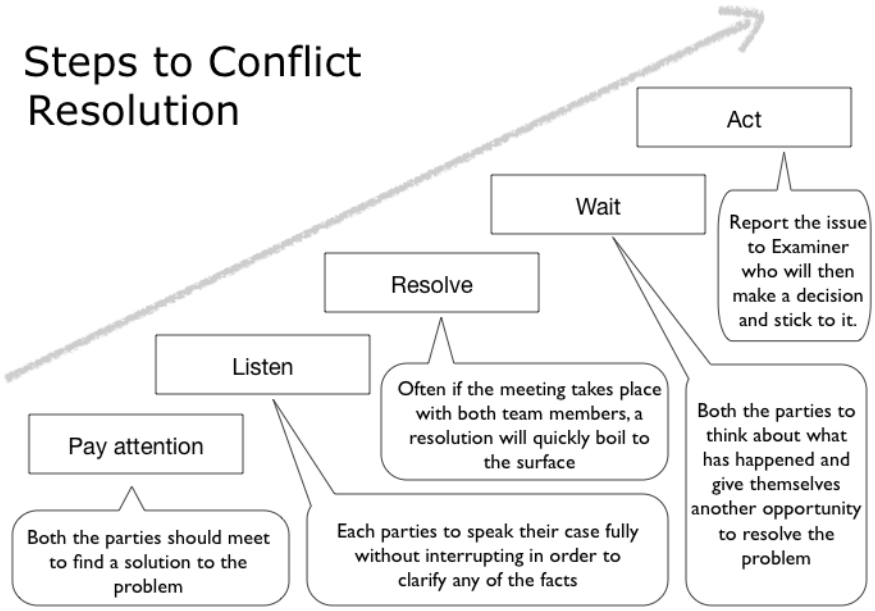

Figure 6: Steps to Conflict Resoution

Suggested Strategy

You have almost three weeks to finish the assignment. Plan to complete it in time. Do not write all of the code in one sitting and expect that everything will be working smoothly magically.

First week Read assignment specification carefully. Clarify anything unclear by putting a post on the Assignment 2 Public Forum. Have your team meeting regularly (face-to-face or on private forum/chatroom) for discussions on assignment requirements and team working style. Think about how to do it, how to test it, devise high-level algorithms for each independent part of the assignment. Begin to type program (with comments), in incremental stages, and help each other in team.

Second week Keep working on team basis and have your team meeting regularly. Re-read the specification, continue to refine design of various sections to code, bring up any problems to team for advice or to the public forum if necessary. By the end of the term you should have had a working program and some integration tests done.

Last week Fully test the program; have another review on the source code; re-read the specification (especially marking criteria) to make sure you have done exactly what is required. Have a team meeting to discussion the experience. Write the "*Statement of Completeness*".