Applied Computing: Software Development  
Unit 3 Area of Study 1

### Programming

School-assessed Coursework (Practice Tasks)

# **VCE Applied Computing: Software Development**

## **Unit 3 Area of Study 1: Programming practice**

On completion of this unit the student should be able to interpret teacher-provided solution requirements and designs, and apply a range of functions and techniques using a programming language to develop and test working software modules.

Students apply computational thinking skills when interpreting given solution requirements and designs, and when developing them into working modules.

**Task conditions**

**Allowed resources:** *your own notebook,*

**Time allocated to this task:** *four class periods*

**Marks allocated:** *100 marks*

## **Outcome brief**

A group of students at Northcote High School are setting up a second-hand shop to buy and sell used electronic devices/parts such as laptops, hard drives, CAS calculators, switches. At the end of the year, older students can sell their used devices to the shop, and the shop will make a small profit when selling them to younger students for the following year.

They have approached you to develop simple software to assist with their shop.

## **General information**

You must complete four tasks of increasing complexity by writing programs to implement the design provided. The designs in all cases are provided for you, however, keep in mind that you must:

demonstrate the use of functions/methods, with parameters and return values, where appropriate.

include rich internal documentation that fully describes your logic and reasoning and implement code layout conventions (e.g. white space, indenting) for readability.

use meaningful names for your project, data structures, functions/methods and variables, adhering, where reasonable, to code conventions for the programming language.

validate input data and user input.

fully and formally test your programs, providing clear evidence of operation.

**Testing**

You should perform testing for each task. For Tasks 2, 3 and 4 this includes constructing appropriate test data based on the example.

A testing table, like the template provided in **Appendix 1**, should be provided as evidence.

**Task 1**

**What are my old electronic devices worth?** (sequence, branching, iteration, variables, functions)

The first program, to be used at the end of a year, allows a student to determine how much they will get for their collection of used devices if they sell it to the shop. It should work as follows:

STEP 1: Prompt the user to input the following for a device:

how old is the device (for how many years has it been used)?

name of the device (optional)

category it belongs to (optional)

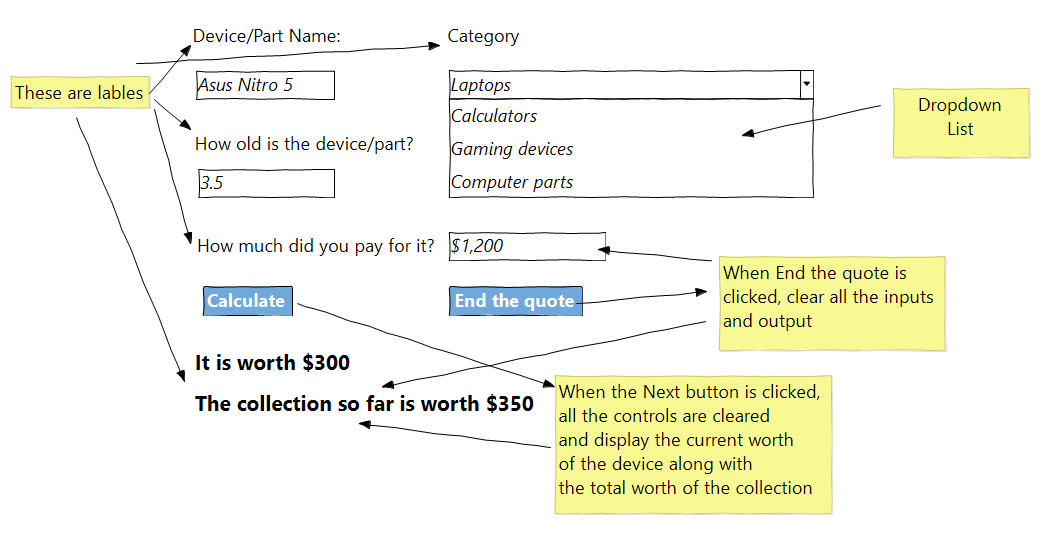
how much did you pay the device?

STEP 2: When the Calculate button is clicked, calculate and display the current value of the device. This is what the shop will pay for it. See the next page for details on how to calculate this value.

STEP 3: Display the **cumulative** value of the collection of devices based on what has already been entered.

STEP 4: The user can enter another device to calculate its current worth and cumulative worth

SETP 5: The user clicks the End the quote to exit the program (all inputs and output are cleared)



*Mock-up of program in use. User input is in italic.*

**The current value of an electronic item is calculated as follows:**

the item loses 20% of its original value when purchased after one year.

across the second year, it is devalued more. It loses 40% of its value when purchased at the start of that year.

across the third year, it will lose 60% of its value from the beginning of the year, and so on.

all devices become worthless after five years.

##### **Pseudocode**

This pseudocode describes a function to perform the calculation:

**Function** calculateCurrentValue(purchasedValue, age)

depreciation ← purchasedValue × 0.2 × age

**If** depreciation > purchasedValue

**Return** 0

**End If**

**Return** purchasedValue - depreciation

**End Function**

## **Task 2**

## **Calculate profit** (reading in CSV file)

It's the end of the year and the shop has bought and sold over 100 items. Now the students running the shop want to know how much profit they made.

The table printed in Appendix 2 gives an example of the data file recording each sale.

The fields (columns) are described as follows:

**item name** – the name of the item that was bought and sold

**category** – it includes laptops, calculators, computer parts, gaming items

**seller** – the name of the student who sold the item to the shop. *(Note: a seller may appear more than once)*

**purchase price** – the value paid by the shop to the original owner

**purchaser** – the name of the student who bought the item from the shop, or 'NA' if the item never got sold *(Note: a purchaser may appear more than once)*

**sale price** – the value paid to the shop by the new owner, or 'NA' if the item never got sold

**rating** – currently unused, but will be used in the last task.

This program should work as follows:

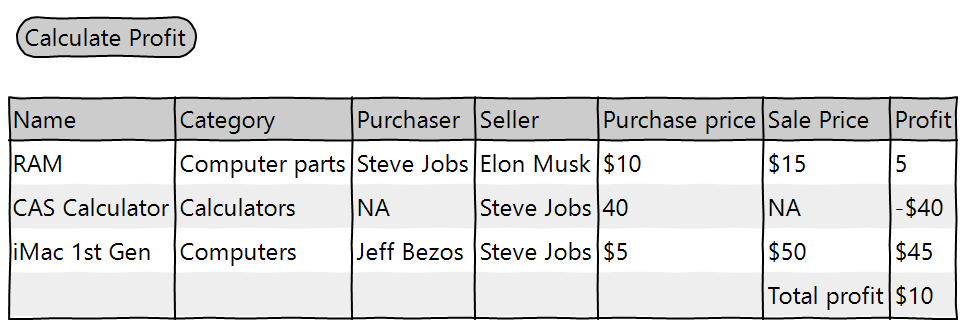
STEP 1: Read in the CSV file row by row, keeping a cumulative total profit:

if the item was sold, calculate the profit for that item by subtracting the **purchase price** from the **sale price**. This value is added to the cumulative total profit.

if the item never got sold, it is a loss and its **purchase price** must be subtracted from the cumulative total profit.

display the item name, purchaser, seller, purchase price, sale price and profit or loss for that item. You can do so by saving the record/row in a one-dimensional array/list and then Add it to the datagridview’s Rows property.

STEP 2: Display the total profit at the end.



Hints: for this task, you do not have to read the rows from the file into a list of objects. Simplely iterate each row in the file and display required information.

<http://csharp.net-informations.com/datagridview/csharp-datagridview-hide-column.htm>

##### **Pseudocode for your reference**

This pseudocode describes the operation of the program (without initialising the data grid view control).

totalProfit ← 0

**For** each line in the csv file

name ← first field from line

purchasePrice ← fourth field from line

salePrice ← fifth field from line

**If** salePrice = 'NA'

profit← purchasePrice× -1

**Else**

profit← salePrice - purchasePrice

**End If**

**totalProfit** ← totalProfit + profit

newRow ← [name,purchasePrice,salePrice,profit]

datagridview.Add(newRow)

**End For**

**Display** 'Total profit is ', totalProfit

## **Task 3**

## **Filtering and sorting the sale data** (sorting and searching algorithms)

Finally, the students would like a simple software solution to help examine their sale data. They'd like options to:

display the data only for a specific category.

sort the data by rating.

You'll start with the same CSV file used in TASKS 2 (also printed as a table in Appendix 2), which is an example of the data file recording each sale.

The fields (columns) are described as follows:

**item name** – the name of the item that was bought and sold

**category** – it includes laptops, calculators, computer parts, gaming items

**seller** – the name of the student who sold the item to the shop. *(Note: a seller may appear more than once)*

**purchase price** – the value paid by the shop to the original owner

**purchaser** – the name of the student who bought the item from the shop, or 'NA' if the item never got sold *(Note: a purchaser may appear more than once)*

**sale price** – the value paid to the shop by the new owner, or 'NA' if the item never got sold

**rating** – a whole number between 1 and 5 representing a star rating, or 'none' if the item hasn't been rated yet.

**This program should work as follows:**

STEP 1: Read in the CSV file *once only*. For each row in the file, **store the values in an object with appropriate property/member names to match the columns. All the rows should be kept in a list.**

STEP 2: Prompt the user to select one of the following filters:

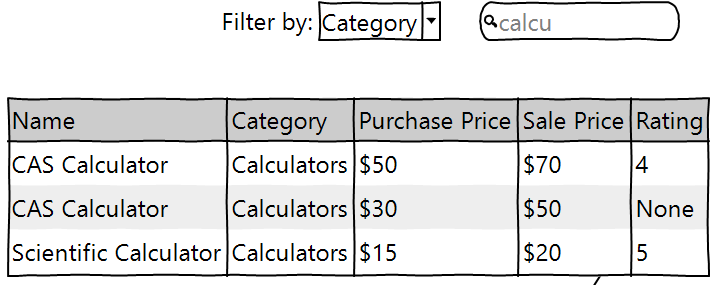
'category',

'rating',

STEP 3: Based on the user's choice:

if the user selected 'category', prompt the user for a category name to use as a filter. Then, display only entries whose **category** contains the response. *Note, an exact match is not required. Part of the text is allowed. eg. ‘calcu’ should bring include all items containing ‘calcu’, case insensitive.*

if the user selected 'rating', display all entries sorted by **Rating** in ascending order, with 'none' entries last.



## **Task 4**

**Rate an item** (writing a CSV file, data structures, searching algorithms)

The students want to set up a kiosk in the shop so that buyers can leave a star rating for the item they bought. (Note, they will rate the state of the item purchased, e.g. wear and tear). The program will allow a user to search for their purchase, then add or replace a star rating.

You'll start with the same CSV file used in Task 2 (also printed as a table in Appendix 2), which is an example of the data file recording each sale.

The fields (columns) are described as follows:

**item name** – the name of the item that was bought and sold

**category** – it includes laptops, calculators, computer parts, gaming items

**seller** – the name of the student who sold the item to the shop. *(Note: a seller may appear more than once)*

**purchase price** – the value paid by the shop to the original owner

**purchaser** – the name of the student who bought the item from the shop, or 'NA' if the item never got sold *(Note: a purchaser may appear more than once)*

**sale price** – the value paid to the shop by the new owner, or 'NA' if the item never got sold

**rating** – a whole number between 1 and 5 representing a star rating, or 'none' if the item hasn't been rated yet.

**This program should work as follows:**

STEP 1: Read in the CSV file *once only*. ***For each row in the file, store the values in an object with appropriate properties to match the columns. All the rows should be kept in a list.***

STEP 2: Prompt the user to enter:

the name of the purchaser

the name of the item

STEP 3: Find the matching record in the list by searching for the name of the item in the Name field *AND* the complete name of the purchaser in the Purchaser field. The search should be case insensitive, but otherwise the text for both fields must be an exact match.

if a matching record is found, display it and go to STEP 4

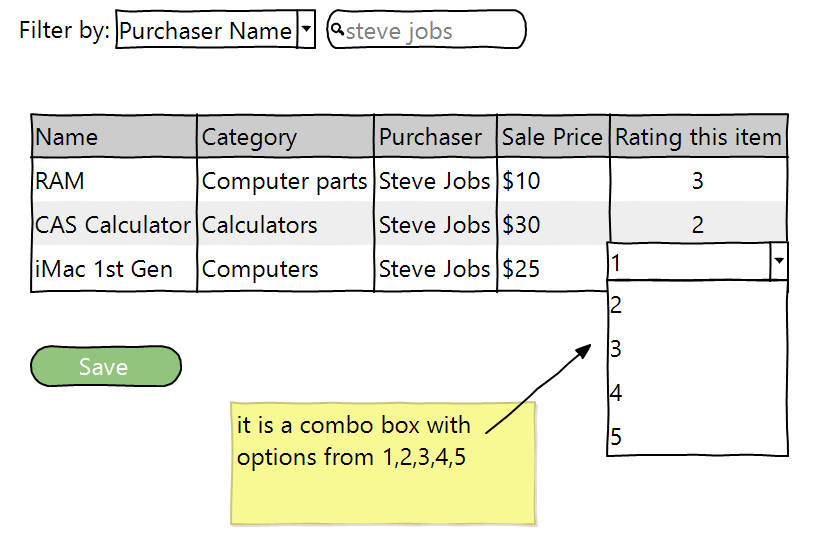
If a matching record is not found, display an appropriate response.

STEP 4: Prompt the user to enter a new star rating from 1 to 5 in the combo box.

STEP 5: Update the record with the new star rating, replacing any existing value for the rating field.

STEP 6: Write all the records into a new output CSV file. The output CSV should be valid CSV, able to be opened in spreadsheet software and using the same columns as the input CSV file.

STEP 7: End the program.



*Mock-up of program in use.*

*Hint: as shown in the mock-up, a combo box is used for entering the rating for the sake of validation. Here is a link to teach yourself how to add a combo box in the data grid view control. You can reference the data using the column index*

[*http://csharp.net-informations.com/datagridview/csharp-datagridview-combobox.htm*](http://csharp.net-informations.com/datagridview/csharp-datagridview-combobox.htm)

*You can choose the appropriate sorting algorithm.*

Appendix 1: Testing table template

Use the following testing table to check your module's functionality. The completed test table should be included with every submission.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Expected result** | **Actual result** | **Pass/fail** | **Remedy** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Action**: some action, event or input to the system which should provoke a response. Typically, this uses a verb: ‘Click the select button’, ‘Select the third item from the drop down box’, ‘Focus a light on the light sensor’.

**Expected response**: this describes what the system *should* do in response to the action taken. It could start with the words ‘The system shall...’. An example might be ‘display the words “HELLO WORLD” in upper text box’. Be specific and definite.

**Actual response**: this describes what *actually* happened (or didn't happen). Try to gather as much information as possible here. ‘The text “HELLO WOR” was displayed and filled the whole text box’.

**Pass/fail**: if the actual result matches the expected result exactly, the test passes.

**Remedy**: if you can, describe what action needs to be taken to fix the problem. This is not always possible at the time of testing but can be added after the program has been fixed. e.g. ‘The size of the text box needs to be increased to accommodate longer sentences.’

Failed test should be redone. A major change or redesign should cause all tests to be re-run as *regression* tests to ensure that no side-effects of a code change have affected other parts of the program.

## **Appendix 2: Example shop data**

This table gives an example of the data kept by the second-hand electronic device shop. A .CSV file with the same content is optionally provided.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Category** | **Seller** | **Purchase price** | **Purchaser** | **Sale price** | **Rating** |
| iMac 1st Gen | Computers | Steve Jobs | 5.00 | Elon Musk | 15.00 | 5 |
| CAS Calculator | Calculators | Marie Curie | 30.00 | Thomas Edison | 50.00 | 4 |
| PS3 Controller | Gaming devices | Ken Kutaragi | 6.50 | NA | NA |  |
| DDR2 RAM 8GB | Computer parts | Bill Gates | 3.75 | Wall-E | 6.25 | 2 |
| USB Drive 16GB | Computer parts | Doctor Who | 5.16 | Homer Simpsons | 7.16 | none |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |