9608/22/PRE/O/N/20

Files

These are the files that constute the solution to the pre-release material for Computer Science component 9608/22 of the October/November 2020 examination series.

Filename	Type	Purpose
9608_w20_PM_22	`.pdf`	The pre-release material file released by CAIE.
Planning	`.md`	This is the markdown text file this PDF was created from.
Planning	`.pdf`	You are currently reading this file. It describes the solution used in answering the pre-release material and houses all material apart from code (such as identifier tables and structured English).
Main Python notebook	`.ipynb`	The Jupyter Notebook in which the Python code was originally written.
Main Python notebook	`.pdf`	The PDF version of the Jupyter Notebook (for the viewer whose system doesn't have Jupyter).
Main Python Code	`.py`	The Python 3.8 file that contains all executable code (for the viewer whose system doesn't have Jupyter).
Main Pseudocode file	`.psu`	The file containing all the pseudocode, written using an open-source custom-built extension.
TASK 1.1	`.png`	Flowchart as required for TASK 1.1.
TASK 2.2	`.png`	Flowchart as required for TASK 2.2.
Item Records	`.txt`	The text file of records as required by TASK 2.

TASK 1 – Algorithms, arrays and pseudocode

> The following four 1D arrays can store different pieces of data about stock items in a shop:

Array Identifier	Data Type
ItemCode	`STRING`
ItemDescription	`STRING`
Price	`REAL`
NumberInStock	`INTEGER`

The `ItemCode` must be of type `INTEGER` and be between `1000` and `9999`. It is, however, stored as type `STRING`.

Task 1.1

> Design an algorithm to search for a specific value in ItemDescription and, if found, to output the array index where the value is found. Output a suitable message if the value is not found.
> Document the algorithm using:
>
> - structured English
> - program flowchart
> - pseudocode.

We will assume that the array `ItemDescription` is already defined and populated as an `ARRAY[1:n] OF STRING` where `n : INTEGER` is the number of items. The program in structured English and the identifier table are given below, and the pseudocode for this step is in the pseudocode file.

Identifier	Data type	Purpose
n	`INTEGER`	Total number of elements (pre-defined)
Index	`INTEGER`	The index number after searching
Counter	`INTEGER`	The running counter for the loop
DesiredValue	`STRING`	The value input from the user
ItemDescription	`ARRAY[1:n] OF STRING`	The pre-populated array of descriptions

Program in Structured English

1. Set `Index` equal to `-1`.
2. Prompt the user for the value they would like to search, and input `DesiredValue`.
3. Traverse `ItemDescription` and check whether any element is equal to `DesiredValue`.
4. If any element is equal, record the its index value in `Index` and break out of the loop. The loop may end before the element was found.
5. If `Index` is equal to `-1`, the element was not found. Output an error message to the user.
6. Otherwise (if `Index` was not equal to `-1`), the element was found. Output the value `Index` in this case.

TASK 1.2

> Consider the difference between algorithms that search for a single or multiple instance of the value.

To search for multiple values, we can consider an array `Indices : ARRAY[1:n] OF INTEGER`, initialized to contain `-1`s, instead of the `Index` value. The program can then record the indices of matches in the array instead of a single value. We can then use a `REPEAT UNTIL` loop to traverse `Indices` and output the indices of matches until it reaches a value if `-1`.

TASK 1.3

> Extend the algorithm from Task 1.1 to output the corresponding values from the other arrays.

Extension is recorded in the pseuducode file. We will assume that the following arrays pre-populated.

Identifier	Data type	Purpose
ItemCode	`ARRAY[1:n] OF STRING`	The pre-populated array of codes for items
Price	`ARRAY[1:n] OF REAL`	The pre-populated array of prices of items
NumberInStock	`ARRAY[1:n] OF INTEGER`	The pre-populated array of the number of items in stock

TASK 1.4

> Write program code to produce a report displaying all the information stored about each item for which the number in stock is below a given level.

This segment will be planned using pseudocode in the pseudocode file. The identifier table is given below.

Identifier	Data type	Purpose
n	`INTEGER`	Total number of elements
ThresholdLevel	`INTEGER`	The minumum stock threshold input to check against
Counter	`INTEGER`	The running counter for the loop

TASK 2 – Programs containing several components

> The stock data described in Task 1 are now to be stored in a text file. Each line of the file will correspond to one stock item.

TASK 2.1

> Define a format in which each line of the text file can store the different pieces of data about one stock item.
> Consider whether there is a requirement for data type conversion.

The values for any given reocrd will be separated by colons. Records themselves will be separated by newline characters.

```
:ItemCode_0:ItemDescription_0:Price_0:NumberInStock_0\n
:ItemCode_1:ItemDescription_1:Price_1:NumberInStock_1\n
:ItemCode_2:ItemDescription_2:Price_2:NumberInStock_2\n
```

Values for `Price` and `NumberInStock` would have to be convetred to `STRING` using `NUM_TO_STRING()` in pseudocode or `str()` in Python because the functions to write only take a `STRING` as an input.

TASK 2.2

> Design an algorithm to input the four pieces of data about a stock item, form a string according to your format design, and write the string to the text file.
> First draw a program flowchart, then write the equivalent pseudocode.

The pseudocode and flowcharts for this section are stored in their repective files. The identifier table is given below here.

Identifier	Data type	Purpose
RecordsFile	`STRING`	Constant to store the name of the file containing the records
NewItemCode	`STRING`	The code of the new item
NewItemDescription	`STRING`	The description of the new item
NewPrice	`REAL`	The price of the new item
NewNumberInStock	`INTEGER`	The number of the new item in stock
WriteString	`STRING`	The string that will be written to the file after concatenation

TASK 2.4

> Consider the different modes when opening a file.

Files can be opened using the following modes in Python.

Mode	Functionality
`READ`	Allows software to read the contents of the file. It cannot alter (append, delete or modify) them.
`WRITE`	Allows the software open a file to write (typically delete or modify; it can append by controlling the pointer) contents only. It cannot read them.
`APPEND`	Allows the software to append a file only. It cannot read, modify or delete.
`READ` and `WRITE`	Allows a software to read and write.
`READ` and `APPEND`	Allows a software to read and append. It cannot modify or delete.

For the tasks given so far, `READ` and `APPEND` are the most appropriate. We are required to read the file to fetch records, and add new records. While `WRITE` also satisfies the latter, we don't need to modify or delete data and would prefer not to risk doing so accidentally.

> Discuss the difference between creating a file and amending the contents.

By its definition, to amend something is to "make changes to" it. Thus, it must exist in advance for us to be able to amend it. In contrase, when we create a file, we don't have anything pre-defined.

> Extend the program to include a menu-driven interface that will perform the following tasks:
>
> 1. Add a new stock item to the text file. Include validation of the different pieces of information as appropriate. For example, item code data may be a fixed format.
> 2. Search for a stock item with a specific item code. Output the other pieces of data together with suitable supporting text.
> 3. Search for all stock items with a specific item description, with output as for task 2.
> 4. Output a list of all stock items with a price greater than a given amount.

The planning for each of these in the pseudocode files. The identifier table is below and shows which program(s) use the identifier.

Identifier	Data type	Purpose	Program(s)
`ExtractDetails()`	`PROCEDURE`	A procedure that extracts the values of each field from a given string and stores them into a given array	2, 3, 4
RecordString	`STRING`	The concatenated string that is passed into the procedure	`ExtractDetails()` scope
SearchString	`STRING`	`RecordString` with a concatenated colon `:` to have a cosistent delimeter	`ExtractDetails()` scope
Details	`ARRAY[1:4] OF STRING`	The array into which all fields are extracted	`ExtractDetails()` scope
Position	`INTEGER`	Pointer to the current position in the string	`ExtractDetails()` scope
Counter	`INTEGER`	The counter for the `FOR` loop iterations	`ExtractDetails()` scope
CurrentChar	`CHAR`	The character currently being examined	`ExtractDetails()` scope
`GetItemCode()`	`FUNCTION`	A function that inputs and returns a valid item code	1, 2
Valid	`BOOLEAN`	Flag for whether or not the the current input is valid	`GetItemCode()` scope
TestItemCode	`INTEGER`	The item code temporarily used for input validation	`GetItemCode()` scope
RecordsFile	`STRING`	Constant to store the name of the file containing the records	1, 2, 3, 4
FileObject	`FILE OBJECT`	Reference to the opened file	1, 2, 3, 4
FileData	`STRING`	Data read from the file while searching	2, 3, 4

Identifier	Data type	Purpose	Program(s)
DetailsOfRecord	`ARRAY[1:4] OF STRING`	Values of all fields of a particular record	2, 3, 4
NewPrice	`REAL`	The price of the new item	1
NewNumberInStock	`INTEGER`	The number of the new item in stock	1
WriteString	`STRING`	The string that will be written to the file after concatenation	1
DesiredItemCode	`INTEGER`	The given item code that will be searched for	2
Found	`BOOLEAN`	Flag for whether or not the desired value is found	2
DesiredItemDescription	`STRING`	The given item description that will be serched for	3
DesiredPrice	`REAL`	The given price that will be searced for	4