**Introductory Programming – Assignment 1**

**TRUR1278 Introductory Programming**

**Word count: 1522**

Abstract

This report highlights the approach taken to problem-solving regarding the vending machine. It goes on to plan and design the logic for the system itself, making use of both pseudocode and flowcharts to illustrate the flow of the program and the algorithms inside it. Next, the implementation of the program itself contains working and fully commented code explaining why certain lines are necessary. Testing follows the implementation with xyz….. Finally, the program is evaluated, showing what went well, where there were problems and what potential improvements could be added with version two.

Contents

[Abstract 2](#_Toc184120190)

[1.0 Introduction 4](#_Toc184120191)

[1.1 Assumptions 4](#_Toc184120192)

[2.0 Problem analysis 5](#_Toc184120193)

[2.1 Understanding the problem 5](#_Toc184120194)

[2.2 Devising a plan 5](#_Toc184120195)

[2.3 Carrying out the plan 6](#_Toc184120196)

[2.4 Looking back 6](#_Toc184120197)

[3.0 Program design 7](#_Toc184120198)

[3.1 Flowcharts 7](#_Toc184120199)

[3.2 Pseudocode 15](#_Toc184120200)

[3.3 Explanation of methods used 15](#_Toc184120201)

[4.0 Implementation 16](#_Toc184120202)

[5.0 Testing 17](#_Toc184120203)

[5.1 Testing evaluation 24](#_Toc184120204)

[6.0 Reflection 25](#_Toc184120205)

[6.1 Meeting the customer brief 25](#_Toc184120206)

[6.2 Version 2 improvements 25](#_Toc184120207)

1.0 Introduction

The goal of this assignment is to produce a working vending machine program that uses all the major programming concepts (functions, loops, conditionals, etc) whilst also carefully planning the program using pseudocode and flowcharts. The program will also need to be thoroughly tested to ensure that everything functions as intended and that errors are handled appropriately.

1.1 Assumptions

Several assumptions have been made for this project. The first assumption that has been made is that the user will want a User Interface (UI) that they can interact with to purchase a drink.

2.0 Problem analysis

There are four stages to problem-solving: Understanding the problem, Devising a plan, Carrying out the plan and Looking back (Polya, 1945). So, before the problem can be analysed, first, the problem must be completely understood.

2.1 Understanding the problem

The problem is as follows - There is a vending machine that only serves one drink: KernowCoke. KernowCoke costs £1.25. The machine only accepts coins in the following denominations: 1p, 2p, 5p, 10p, 20p, 50p, £1, £2. A customer can insert multiple coins to pay for the drink. The machine should prompt the user for more coins until they have paid for the drink. The machine needs to calculate and then show how much change is due and what coins to dispense.

The machine must also be able to handle potential erroneous inputs and deal with them appropriately. For example, trying to insert a coin that does not exist should re-prompt the user for a valid coin insertion.

2.2 Devising a plan

Now that the problem is fully understood, a plan to tackle it can be implemented. Firstly, some guidelines must be followed for the plan: the code must make use of functions to ensure neat organisation, the most suitable data types should be used, such as integers for calculations that involve pennies, a loop should be used during coin insertion to make sure that the correct amount is received, and invalid inputs must be handled.

In line with the guidelines, there will be separate functions for each of the core steps of the solution. Where possible, a function will only perform one task before returning a value. These functions will also have self-identifying names to ensure maximum readability.

A flowchart and pseudocode need to be developed to demonstrate how the logical aspects of the program will execute. These tools will provide the foundations of the live code implementation since most of the logic will just need to be translated from these mediums into real code.

Finally, what happens with edge cases needs to be considered. If an erroneous input is detected, such as trying to insert a 3p coin, the system needs to recognise that this is invalid and deal with it appropriately, e.g. reject it and ask for a real coin.

2.3 Carrying out the plan

See 4.0 Implementation for an analysis of the core functions of the code and why each.

2.4 Looking back

See 6.0 Reflection for details on how the project went and what could be improved.

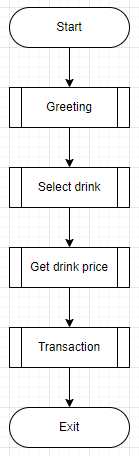
3.0 Program design

Before the program is implemented, the design has been created using both pseudocode and flowcharts to better understand the logical flow of the program.

3.1 Flowcharts

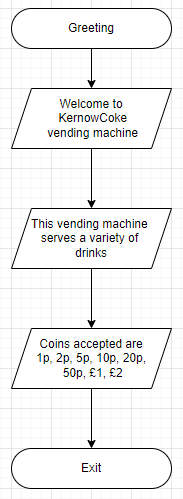
*“the flowchart serves as the central design document around which systems analysts, computer programmers, and end users communicate, negotiate, and represent complexity”* (Ensmenger, 2016)

For this project, the flowcharts will form the basis for the code implementation. Whilst there may be some adaptations required to properly implement the flowcharts, the basic logic will remain the same. It is also quicker to analyse an algorithm using flowcharts compared to pseudocode (Andrzejewska and Stolińska, 2022). Flowcharts can also serve as documentation to help developers better understand the logic of the program during future maintenance.

**Main:**

Main calls all the functions necessary for program execution.

**Greeting:**

****Greeting prints out all the info that customers need to see before making a purchase.

**Insert coins:**

**A diagram of a coin

Description automatically generated**Coin insertion utilises a loop to handle both multiple coin inputs as well as erroneous inputs.

**Format money:**

**A diagram of a money flow

Description automatically generated**

Format money takes an integer penny value and either converts it to pounds and adds a pound symbol if the value is more than 99p or adds a p to the end if it is less than £1.

**Transaction:**

**A diagram of a flowchart

Description automatically generated**

Transaction takes care of the whole payment, including dispensing the correct amount of change. It calls both the insert coin and calculate change functions.

**Calculate change:**

**A diagram of a flowchart

Description automatically generated**Calculate change works out what coins to dispense to the customer. It is only called when an overpayment is made.

**A diagram of a money transfer

Description automatically generated**

**Drink selection:**

**A diagram of a drink

Description automatically generated**

Drink selection presents a list of available drinks and then asks the user to select one. It utilises a loop to handle erroneous choices, allowing customers to retry if they enter an invalid drink.

**Drink price:**

A diagram of a drink

Description automatically generatedCompares drink price against dictionary containing prices, then returns the prices in integer form in pennies.

3.2 Pseudocode

Pseudocode provides further design documentation on how the program will be implemented. Pseudocode is a more technical document than flowcharts as it is purely text-based. Pseudocode looks much more like what the final implementation will look like using actual code.

3.3 Explanation of methods used

Whilst this program itself does not have any form of database vulnerable to attack, it is still good practice to ensure input validation. If this application were to be used in a real vending machine, it could be faced with bad actors. These bad actors may try to input ‘Bad data’, which *“can lead to the exploitation of security vulnerabilities in the system”* (Miller, 2014). In the case of the vending machine, bad actors may try to get drinks without paying or force the machine to give out more change than expected. Where the program requires user input, loops have been used so that erroneous inputs force the code to execute again until a valid input has been received. For coin insertion specifically, the use of a loop also allows for multiple coins to be inserted in a row, meeting another guideline.

Functions have been used to separate the logical components of the code. This modularity makes it easier to read and easier to maintain in future (Martínez and Pardo, 2013), which is beneficial to developers who may not have worked on the original project. The developers can look at each function and quickly work out what each does and how it interacts with the rest of the program.

Appropriate data types have been used in the program. Floating points should not be used where the amount of money needs to be exact due to their representational inaccuracies (Dale and Weems, 1998). As such, all money calculations are carried out in pennies.

4.0 Implementation

The program was implemented entirely using Python. Python is advantageous because it does not need a user interface to run the program; the inputs and outputs can all occur in the terminal. Python also has a simple syntax, making development quicker than other languages.

5.0 Testing

**Test Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Purpose of Test** | **Expected Outcome** | **Test Data** |
| 1 | Does the program run without crashing | Program runs without crashing | n/a |
| 2 | Can you select a drink with expected input | Drink selected and transaction started | Choice: 1A |
| 3 | Can you select a drink with correct code but with the wrong case | Drink is still selected and transaction started | Choice: 1b |
| 4 | Is an erroneous drink choice handled correctly | Error message displayed and drink select prompt reappears | Choice: TruroTango |
| 5 | Does inserting pounds take the correct amount from the total | £1 is taken from the outstanding total and loops back to coin input, leaving £1 outstanding | Choice: 2A. Coin: £1 |
| 6 | Does inserting pennies take the correct amount from the total | 50p is taken from the outstanding total and loops back to coin input, leaving £1.25 outstanding | Choice: 2B. Coin: 50p |
| 7 | Does inserting pounds in without the £ symbol ask customer to insert a valid coin | Displays an error message asking the customer to insert a valid coin | Choice: 1A. Coin: 1 |
| 8 | Does the program allow several sequential coin insertions | Loop will continue to ask for coins until the drink is paid off | Choice: 2A. Coins: £1, 50p, 50p |
| 9 | Does the program calculate the correct change when the exact amount is given | No change will be given since £1 is the exact price | Choice: 1B. Coin: £1 |
| 10 | Does the program calculate the correct change when an overpayment is made. | Change will be given: 50p, 20p, 5p | Choice: 1A. Coin: £2 |
| 11 | Is an erroneous coin insertion handled correctly | Displays an error message asking the customer to insert a valid coin | Choice: 2B. Coin: £10 |

**Test Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 1 | **Purpose of Test:** | Does the program run without crashing |
| **Test Data:** | n/a | | |
| **Expected Outcome:** | Program runs without crashing | | |
| **Actual Outcome:** | Program runs without crashing | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
| **A screenshot of a menu  Description automatically generated** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 2 | **Purpose of Test:** | Can you select a drink with expected input |
| **Test Data:** | Choice: 1A | | |
| **Expected Outcome:** | Drink selected and transaction started | | |
| **Actual Outcome:** | Drink selected and transaction started | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
| **A screenshot of a computer  Description automatically generated** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 3 | **Purpose of Test:** | Can you select a drink with correct code but with the wrong case |
| **Test Data:** | Choice: 1b | | |
| **Expected Outcome:** | Drink is still selected, and transaction started | | |
| **Actual Outcome:** | Drink is still selected, and transaction started | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |
| **Test No.:** | 4 | **Purpose of Test:** | Is an erroneous drink choice handled correctly |
| **Test Data:** | Choice: TruroTango | | |
| **Expected Outcome:** | Error message displayed and drink select prompt reappears | | |
| **Actual Outcome:** | Error message displayed and drink select prompt reappears | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 5 | **Purpose of Test:** | Does inserting pounds take the correct amount from the total |
| **Test Data:** | Choice: 2A. Coin: £1 | | |
| **Expected Outcome:** | £1 is taken from the outstanding total and loops back to coin input, leaving £1 outstanding | | |
| **Actual Outcome:** | £1 is taken from the outstanding total and loops back to coin input, leaving £1 outstanding | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 6 | **Purpose of Test:** | Does inserting pennies take the correct amount from the total |
| **Test Data:** | Choice: 2B. Coin: 50p | | |
| **Expected Outcome:** | 50p is taken from the outstanding total and loops back to coin input, leaving £1.25 outstanding | | |
| **Actual Outcome:** | 50p is taken from the outstanding total and loops back to coin input, leaving £1.25 outstanding | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 7 | **Purpose of Test:** | Does inserting pounds in without the £ symbol ask customer to insert a valid coin |
| **Test Data:** | Choice: 1A. Coin: 1 | | |
| **Expected Outcome:** | Displays an error message asking the customer to insert a valid coin | | |
| **Actual Outcome:** | Program crashes | | |
| **Pass/Fail:** | Fail | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 8 | **Purpose of Test:** | Does the program allow several sequential coin insertions |
| **Test Data:** | Choice: 2A. Coins: £1, 50p, 50p | | |
| **Expected Outcome:** | Loop will continue to ask for coins until the drink is paid off | | |
| **Actual Outcome:** | Loop continues to ask for coins until the drink is paid off | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 9 | **Purpose of Test:** | Does the program calculate the correct change when the exact amount is given |
| **Test Data:** | Choice: 1B. Coin: £1 | | |
| **Expected Outcome:** | No change will be given since £1 is the exact price | | |
| **Actual Outcome:** | No change is given since £1 is the exact price | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 10 | **Purpose of Test:** | Does the program calculate the correct change when an overpayment is made. |
| **Test Data:** | Choice: 1A. Coin: £2 | | |
| **Expected Outcome:** | Change will be given: 50p, 20p, 5p | | |
| **Actual Outcome:** | Change is given: 50p, 20p, 5p | | |
| **Pass/Fail:** | Pass | | |
| **Evidence:** | | | |
|  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.:** | 11 | **Purpose of Test:** | Is an erroneous coin insertion handled correctly |
| **Test Data:** | Choice: 1A. Coin: £10 | | |
| **Expected Outcome:** | Displays an error message asking the customer to insert a valid coin | | |
| **Actual Outcome:** | Program crashes | | |
| **Pass/Fail:** | Fail | | |
| **Evidence:** | | | |
|  | | | |

5.1 Testing evaluation

9/11 tests passed without issue. It is good to see that the program functions as expected when input is normal. However, two of the tests did not pass as the program crashed during the two tests. The cause of these errors was initially unclear, with the first theory being that because 1 and £1 were part of the valid strings, it thought they were valid and subsequently tried to continue with data of the wrong type. Upon closer inspection of the code, though, it was clear that the cause of the errors was incorrect syntax. Where there should have been a continue statement on line 30, a break statement was present instead. This caused the program to stop looping, and therefore, there was no coin input to carry out calculations with, causing the crash.

Overall, the testing can be considered a success as it uncovered a key bug in the code that has since been fixed, as well as proving that the code works with all expected inputs.

6.0 Reflection

6.1 Meeting the customer brief

All the basic requirements have been met: The vending machine serves KernowCoke, accepts all the UK coin denominations, allows multiple coins to be inserted for one purchase, calculates the change and displays what coins will be dispensed, and prompts the customer to insert more coins until the drink has been paid off.

In addition to meeting all the basic requirements, one of the optional features has been added, allowing the user to select one of many drink options with different prices.

Overall, this program sufficiently meets the customer brief, though several improvements could be made to improve functionality. These features are discussed below.

6.2 Version 2 improvements

Version 2 of the vending machine application could have many additional features to improve functionality.

The biggest benefit would be a better user interface. Currently, all inputs and outputs occur within the terminal. This can be intimidating to people who are less confident with technology. In a real-life implementation of the machine, a number pad of tactile buttons or a touch screen would make the machine much more accessible to people of all technological ability levels.

Accepting card transactions would make the vending machine more accessible to people who do not carry cash. This would be a complicated implementation, though, as it would need to adhere to GDPR legislation and utilise secure encryption techniques.