UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS132 - Imperative Programming

Practical 6 Specifications

Release Date: 31-03-2025 at 06:00

Due Date: 04-04-2025 at 23:59

Late Deadline: 06-04-2025 at 23:59

Total Marks: 235

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you begin coding.***
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 48 hours after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

In this practical, you will be required to implement a series of decisions using the selection statements that was covered in class and in the study units. You will also be required to implement a single repetition structure based on a flowchart.

For this practical, you will be required to submit flowcharts for the selected function as part of your submission. These files are required for you to be marked on FitchFork.

3 Background

After your successful completion of the recent semester tests, you were approached by a brand new bubble tea shop opening near campus called *BubStop*.

Bubble tea (also called boba tea) is a popular Taiwanese drink made with tea, milk (or fruit flavors), sweetener, and chewy tapioca pearls (boba). It can be served hot or cold and comes in many flavours like classic milk tea, taro, matcha, and fruit-infused varieties. The fun, chewy texture of the pearls makes them a unique and tasty treat!

You have been tasked with creating the software component for *BubStop*'s point of sales (POS) system. A POS system is a combination of hardware and software that processes transactions, manages sales, and tracks inventory for businesses. *BubStop*'s POS will be a terminal-based system, where the user will interact with the system using terminal input and output.

BubStop currently only has five teas, five bubble flavours, and five extras that the client can choose from, but the client can combine and even repeat different teas, bubbles, and extras. (As long as they can pay for it ;))

4 Your Task:

You are required to implement all of the functions in the `BubbleTeaShop` files. You have been provided with a `PriceList.h` file and `Menus.h` and `Menus.cpp` files. Do not modify the provided files as this will negatively impact your submission.

Task	Marks
1_Tea_Tester	36
2_Bubble_Tester	36
3_Extras_Testers	36
4_ResetProcessBill	46
5_FormProcessOrder	63
Testing	18

Table 1: Marks for each task

4.1 PriceList

You will notice that inside the `PriceList` file, there are three different namespaces, namely: `Teas`, `Bubbles`, and `Extras`. Each namespace contains 5 items, which are represented by a string, which is the item's name and the price of the item. Hint: Remember to access a certain constant inside the namespace using:

`namaspaceName::variable`

For example, to access `blackTeaStr` use:

```
Teas::blackTeaStr
```

The `PriceList` file also includes some helpful values, like the currency symbol and the VAT. Note that as the provided files are overwritten on FitchFork, you cannot assume that the prices, currency symbol and VAT will remain constant. As such, **do not** hardcode these values in your functions.

4.2 Menus

This file implements the menus for *BubStop* as terminal based menus. A series of options is displayed to the user and they will need to enter the number that corresponds with the item on the menu they would like to select. For example, consider the following menu:

```
Please select one of the following options for your drink
1 - Add a tea
2 - Add a bubble flavour
3 - Add an extra
0 - Bill please!
```

If the user would like to add a tea, they will enter `1` followed by the `enter key`. If the user would like to add a bubble, they will enter `2` followed by the `enter key`, and so on.

4.2.1 Functions

The following functions have been provided for you:

- `displayWelcome`
  - This function displays the welcome message.
- `displayMainMenu`
  - This function displays the main menu.
- `displayTeaMenus`
  - This function displays the team menu.
- `displayBubbleMenu`
  - This function displays the bubble menu.
- `displayExtraMenu`
  - This function displays the extras menu.
- `displayGoodBye`
  - This function displays the goodbye message.
- `displayInvalidOption`
  - This function displays an error message that the provided parameter was an invalid option.

- `getMenuOption`
  - This function obtains the user input and returns it as an integer.
  - If the input is not able to easily be converted to a non-integer value, the function will return -1.

4.3 BubbleTeaShop

This file contains all of the functions that you need to implement. The file also contains the `Order` namespace, which is used to capture and record an order.

4.3.1 Order

This namespace contains the fields required to capture and calculate an order. When adding more tea flavours to the order, you will need to append[1] the new flavour to the `teas` variable. You will also need to add the tea's price to the `teaSubtotal`. The same concept is used for the bubbles and the extras.

4.3.2 Functions

For all the functions containing the word "process" in the name of the function, it can be assumed that the range of available options is fixed, and can/will never change. You are required to implement the following functions:

- `resetOrder`
  - This function is used to reset the `Order` to the following default values:
    - Any `string` variable inside the `Order`, needs to be set to an empty string.
    - Any `float` variable inside the `Order`, needs to be set to 0.
- `processBill`
  - This function needs to create the bill and display it to the user.

[1]Append means add to the rear.

– The format of the output is as follows:

```
Your bill is:

Teas:
<<All teas in the order>>

Tea total: <<currency symbol>><<total for teas>>

Bubbles:
<<All bubbles in the order>>

Bubble total: <<currency symbol>><<total for bubbles>>

Extras:
<<All extras in the order>>

Extra total: <<currency symbol>><<total for extras>>

Subtotal: <<currency symbol>><<total for teas, bubbles, and extras summed
   together>>
Vat value: <<currency symbol>><<calculated VAT based on the subtotal>>
Total: <<currency symbol>><<subtotal summed with the VAT>>
```

– Where << and >> indicate that this is a description of what should be printed.

– An example of the output, using a currency symbol of `R`, a VAT of 15%, and that each tea is `45`, we obtain:

```
Your bill is:

Teas:
Lemon Tea
Mango Tea

Tea total: R90

Bubbles:
Blueberry Bubble

Bubble total: R8

Extras:
Almond Pearls

Extra total: R5

Subtotal: R103
Vat value: R15.45
Total: R118.45
```

- `getTeaName`
  - This function should obtain the tea that is associated with the passed in parameter, and return the name.
  - The following table contains the association between number and tea:

Tea Number	Tea Name
1	Black
2	Passion Fruit
3	Lemon
4	Mango
5	Honey Lemon

Table 2: Tea Number and the associated Tea

  - Remember to use the string constant for the tea from the `PriceList` file.
  - If the number is invalid, return the following string: "Unknown Tea".
  - You are required to also submit a flowchart representation of this function with the name: `getTeaName.png`
- `getTeaPrice`
  - This function should obtain the tea's price for the tea whose string matches the passed in parameter.
  - For example, if black tea's string is passed in, then the function should return the price of black tea.
  - If the passed-in string does not match any of the teas' strings, then return a price of 0.
  - You are required to also submit a flowchart representation of this function with the name: `getTeaPrice.png`
- `processTeaOption`
  - This function needs to determine which tea the user would like, and add it to the `Order`.
  - The menu number the user entered is passed in as the function's parameter.
  - If the passed-in value is 0, the function should just return `true`.
  - If the passed-in value is valid (inside the range of available tea numbers), the function should add the tea (associated with the passed-in value) string and price to the appropriate variables in `Order`, and return `true`.
  - When adding the string to the appropriate variable in `Order`, add a newline character after adding the tea.
    - For example, it should be:

```
TeaA\nTeaB\nTeaC\n
```

    - If we were to print out the appropriate variable in `Order`, we would see:

```
Lemon Tea
Black Tea
```

  - If the passed-in value is invalid (outside the range of available tea numbers), the function should print out an invalid option error message, and return `false`.
  - You are required to also submit a flowchart representation of this function with the name: `processTeaOption.png`

- `getBubbleName`
  - This function should obtain the bubble that is associated with the passed in parameter, and return the name.
  - The following table contains the association between number and bubble:

Bubble Number	Bubble Name
1	Litchi
2	Strawberry
3	Pomegranate
4	Blueberry
5	Passion Fruit

Table 3: Bubble Number and the associated Bubble

  - Remember to use the string constant for the bubble from the `PriceList` file.
  - If the number is invalid, return the following string: "Unknown Bubble".

- `getBubblePrice`
  - This function should obtain the bubble's price for the bubble whose string matches the passed in parameter.
  - For example, if litchi bubble's string is passed in, then the function should return the price of litchi bubble.
  - If the passed-in string does not match any of the bubbles' strings, then return a price of 0.

- `processBubbleOption`
  - This function needs to determine which bubble the user would like, and add it to the `Order`.
  - The menu number the user entered is passed in as the function's parameter.
  - If the passed-in value is 0, the function should just return `true`.
  - If the passed-in value is valid (inside the range of available bubble numbers), the function should add the bubble (associated with the passed-in value) string and price to the appropriate variables in `Order`, and return `true`.

- When adding the string to the appropriate variable in `Order`, add a newline character after adding the bubble.
  - For example, it should be:

    ```
    BubbleA\nBubbleB\nBubbleC\n
    ```

  - If we were to print out the appropriate variable in `Order`, we would see:

    ```
    Litchi
    Pomegrante
    ```

- If the passed-in value is invalid (outside the range of available bubble numbers), the function should print out an invalid option error message, and return `false`.

- `getExtraName`
  - This function should obtain the extra that is associated with the passed-in parameter, and return the name.
  - The following table contains the association between number and extra:

Extra Number	Extra Name
1	Mango Slush
2	Coffee Slush
3	Extra Ice Slush
4	Fresh Taro
5	Almond Pearls

Table 4: Extra Number and the associated Extra

  - Remember to use the string constant for the extra from the `PriceList` file.
  - If the number is invalid, return the following string: "Unknown Extra".

- `getExtraPrice`
  - This function should obtain the extra's price for the extra whose string matches the passed in parameter.
  - For example, if mango slush's extra string is passed in, then the function should return the price of mango slush extra.
  - If the passed-in string does not match any of the extras' strings, then return a price of 0.

- `processExtraOption`
  - This function needs to determine which extra the user would like, and add it to the `Order`.
  - The menu number the user entered is passed in as the function's parameter.
  - If the passed-in value is 0, the function should just return `true`.

  - If the passed-in value is valid (inside the range of available extra numbers), the function should add the extra (associated with the passed-in value) string and price to the appropriate variables in `Order`, and return `true`.
  - When adding the string to the appropriate variable in `Order`, add a newline character after adding the tea.
    * So, for example, it should be:

      ```
      ExtraA\nExtraB\nExtraC\n
      ```

    * If we were to print out the appropriate variable in `Order`, we would see:

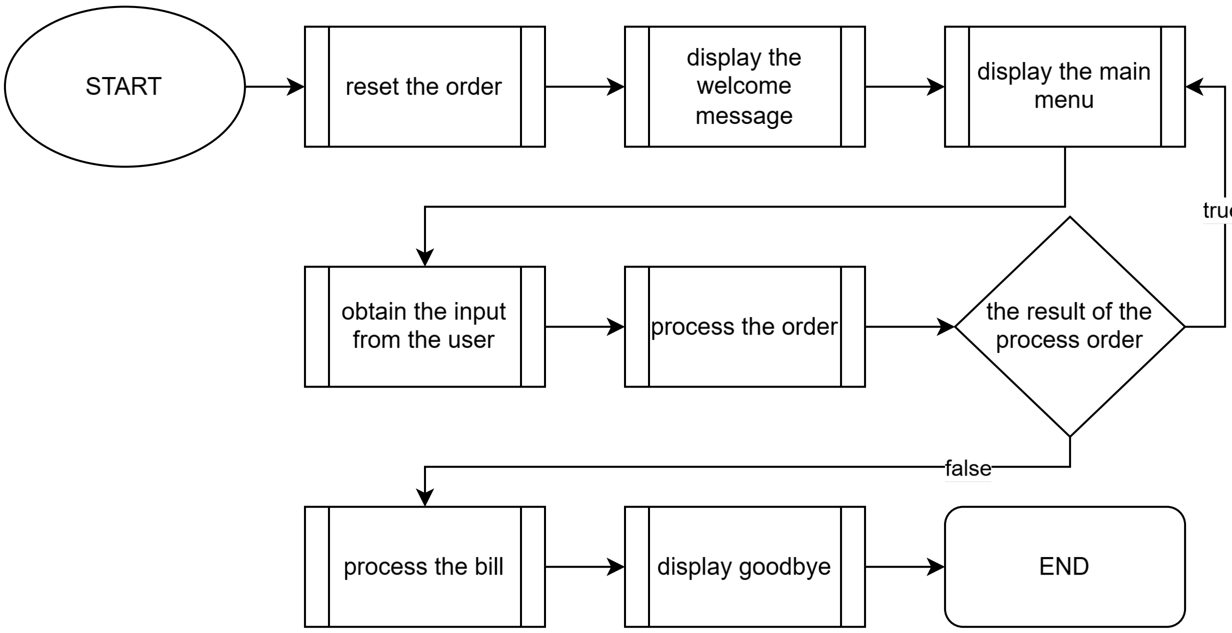
      ```
      Mango Slush
      Extra Ice
      ```

  - If the passed-in value is invalid (outside the range of available extra numbers), the function should print out an invalid option error message, and return `false`.

- `processOrder`
  - This function needs to determine which submenu the user needs to process based on the passed-in parameter.
  - If the parameter value is 0, the function should just return false.
  - If the parameter value is 1, the function should:
    1. Display the tea menu
    2. Obtain the user's selected choice from the tea menu.
    3. Process the user's input for tea using the appropriate function.
    4. If the result of the function call in step 3 is false, display an invalid option error message.
  - If the parameter value is 2, the function should:
    1. Display the bubble menu
    2. Obtain the user's selected choice from the bubble menu.
    3. Process the user's input for bubble using the appropriate function.
    4. If the result of the function call in step 3 is false, display an invalid option error message.
  - If the parameter value is 3, the function should:
    1. Display the extras menu
    2. Obtain the user's selected choice from the extras menu.
    3. Process the user's input for extras using the appropriate function.
    4. If the result of the function call in step 3 is false, display an invalid option error message.
  - Lastly, at the end, the function should return true.

  - You are required to also submit a flowchart representation of this function with the name: `processOrder.png`

- `formOrder`
  - The form order is the "main" loop for the ordering process, and can be described by the following flowchart.
  - You are welcome to use this flowchart as an example of how to draw your own flowcharts[2].



5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [3] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

[2]The flowchart was created using DrawIO
[3]For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

The mark you will receive for the testing coverage task is determined using Table 5:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 5: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Tutor Assistance

For the functions that require a flowchart, tutors have been instructed to only assist them if you can produce a valid flowchart for that function.

7 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **C++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - string
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

8 Upload Checklist

The following C++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- `BubbleTeaShop.cpp`
- `Menus.cpp`
- getTeaName.png
- getTeaPrice.png
- processTeaOption.png
- processOrder.png
- `main.cpp`
- Any textfiles used by your `main.cpp`

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

9 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -Werror -Wall -std=c++98 -g *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 20 submissions and your final submission's mark will be your final mark. Upload your archive to the Practical 6 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**