Silog Vending Machine

Vending or "automatic retailing" has its roots traced back as far as 215BC. However it was only in the early 1880s that the first commercial coin-operated vending machines were introduced in London, England which dispensed post cards. Now, modern vending machines offer more variety of products ranging from drinks, tickets, snacks, cigars, WIFI connections, and many more. In Japan, vending machines are designed to function during blackouts, particularly in the wake of earthquakes and aftershocks, thus utilizing vending machines as an emergency aid.

In an effort to invoke appreciation of the many fascinating automation wonders of our modern era, you will re-create a vending machine. Specifically, you will create a program **that simulates the interaction of a user with a vending machine for silog**. It will handle complete functionalities from start to end of the buying process. The project's basic requirement will be to produce these interactions in a text based menu format. The interface should address limitations of the software simulation such as the "receiving" of the product and "getting" of money from user.

REQUIRED Features and Functionalities

The Main Menu consists of Silog Vending Features, Maintenance Features, and Shutdown Machine. The programmer may choose to require integer like 1, 2, 3 or character like V, M, S for the input menu option. The program proceeds based on options chosen until the user chooses to Shutdown Machine. Please refer to details below:

1. Silog Vending Features

Assume that users of the following features are the customers who will buy silog. In this phase, the following features are done sequentially (i.e., the user need not press any option to View Item, Accept Money, etc). The process already follows these steps: View Items, Accept User Money, Select Item/s, View Total, Confirm or Cancel Transaction, Get Change, and [if confirmed] Get Product. After this the user is asked Start Vending Again? If yes, the whole process starts over again. If no, the user is brought back to the main menu.

a. View Items

The items and their default price in the vending machine are hotdog (PhP9.50), longganisa (PhP20.75), bacon (PhP 12), sausage (PhP35), tapa (PhP22.50), tocino (PhP18), rice (PhP15), and egg (PhP8). The user must be able to view the following details on each item:

- Item Number (number the customer will "press"/type to signify choice)
- Item Name
- Price
- Stock Left (number of available servings. For example 30 grams of bacon is 1 serving, the stocks left is still 1 (not the 30).

This set of information must be updated for every transaction completed.
There must be an indicator that an item is no longer available for purchase.

b. Accept User Money

Denominations allowed:

- Bills – 20 Php, 50 Php, 100 Php, 200 Php, 500 Php
- Coins – 1 Php, 5 Php, 10 Php, 25 Cents, 10 Cents, 5 Cents

To simulate this, the vending machine asks the user what **denomination** is inserted. This goes on until the user says he is done inserting money.

c. Select Item/s

User enters the item number. User can add as many add-ons to his basic silog (1 serving of sinangag and 1 itlog) as he wants in one transaction. If money is not enough, the program notifies the user of lack of money and asks whether to put in more money or cancel the last item selected. Note that each transaction will produce only 1 silog meal, but in this option, the user may choose items that is still available in the machine. For example, if the user chooses the following items as add-on to the basic silog: hotdog, rice, hotdog, tapa; then this means that there are 2 hotdogs, 1 (serving of) tapa, **2** rice, and 1 egg in his 1 order of silog. [There are 2 rice, because the silog already comes with rice and egg, so selecting it in the items mean an additional rice.] There must be at least 1 item selected.

d. View Total and Confirm Transaction

The program displays the total cost and asks for confirmation of the transaction. Once confirmed, the machine then "prepares" the silog for dispensing.

e. Cancel Transaction

The user can cancel the transaction any time before the confirmation is given. Whatever amount given will be returned following the Get Change. Thus, it is possible that the returned denominations are not the same as those that this customer inserted, since smallest number of bills and coins will be the basis.

f. Get Change

Upon confirmation, the program shows the amount of change and **the actual number of each denomination coins/bills given**. Make sure to give the smallest number of bills and coins, except when you run out of some of the needed bill / coin. **Note that the money inserted into the machine as payment may also be used as change**.

g. Get Product

In a text based format, the program shows a message "Get silog from tray bin."

h. Back to Main Menu

This option ends the vending features and goes back to the main menu.

2. Maintenance Features

Assume that users of the following features should be the seller or the owner of the vending machine. Upon entry to this phase, the user is asked to input the password. If correct, the user is given the choice among Inventory Features, Cash Register Features, or Back to Main Menu. If incorrect password, show an error message, then go back to the main menu.

a. Input Password for Seller

Upon entry to the Maintenance Features from the main menu, a numeric security password is required to access any of the features of the Vending Maintenance. For simplicity, let us use 123456 as the password.

b. Inventory Features

i. View Inventory

In this feature, all information of the items should be displayed. Essentially, this is supposed to behave like the View Items option as indicated in the previous page.

ii. Set/Modify Price

In this feature, it will first display all the information about the items. Then, it will ask the user to input the item number whose price will be set or modified. If the item number is valid, then the new price is asked from the user. Note that a negative price is not allowed. Going back to the Vending Features would mean that the prices should already be updated. Saving the updated price for the next run of the program is not expected as a basic requirement.

iii. Stock/Restock Inventory

Once this option is chosen, the program will first display all the information about the items. Then, it will ask the user to input the item number that will be stocked. If there is such an item number, the quantity is asked from the user. This quantity is **added** to the current quantity of the item. Note that a negative quantity is not allowed.

iv. Back to Vending Maintenance Features

This exits the Inventory Features menu and goes back to the Vending Maintenance menu options.

c. Cash Register Features

i. View Cash Register

In this feature, the number of each denomination of bills and coins at hand should be displayed. The total amount is also displayed.

ii. Stock/Restock Cash Register

Once this option is chosen, the program will first display all the information in the cash register. Then, it will ask the user to input the denomination that will be stocked. If it is a valid denomination, the quantity is asked from the user. This quantity is added to the current quantity of the item. Note that a negative quantity is not allowed.

iii. Cash Out

Once this option is chosen, the program will first display all the information in the cash register. Then, it will ask the user if he wishes to input amount of money to claim or to input denomination and quantity. Should the user opt to input the amount he wants to claim from the cash register, the program will then compute and display the total amount of each denomination of bills and coins that will total to the given amount (if these exist). For example, the user wants to claim 38 pesos, the program should display that it is dispensing 1 - PhP20, 1 – PhP10, 1 – PhP5, and 3 – PhP1. However, if the machine does not

have any more PhP10, then the program should display that it is dispensing 1 - PhP20, 3 – PhP5, and 3 – PhP1 (assuming it has enough PhP5). Furthermore, if the machine will not be able to dispense the said amount because of lack of denominations, e.g., only 2 PhP1, then it should have displayed a message "Unable to dispense exact stated amount". Note that this is actually similar in process to the dispensing of change.

On the other hand, if the user chooses to enter the denomination and the quantity, the program must make sure that the denomination is valid and that there is such a quantity that can be claimed/withdrawn before the cash out can be completed.

iv. Back to Maintenance Features

This exits the Inventory Features menu and goes back to the Vending Maintenance menu options.

c. Back to Main Menu

This option ends the maintenance features and goes back to the main menu.

3. Shut Down Machine

Should the user choose this option (from the main menu), the program should first validate that the user is allowed to shut down the machine. It is assumed that the seller or owner of the machine can shut it down. Thus, a numeric password should be keyed in.

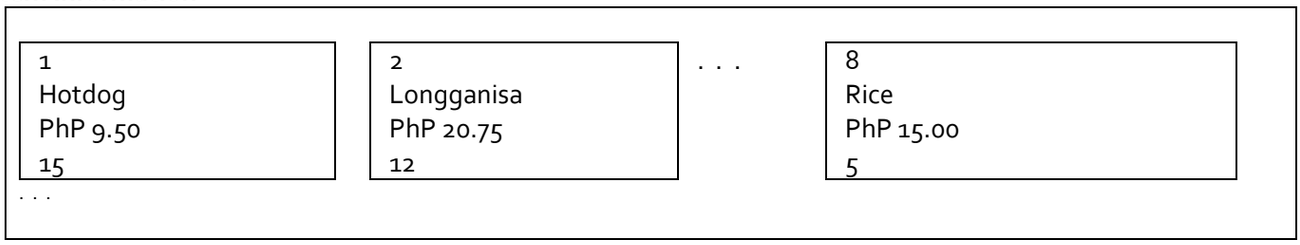
Input Password for Seller

A numeric security password is required to access this functionality. The program asks for the password. If the password is correct, the machine produces a message "Going Offline", then ends the program. Again, for simplicity, we will use 123456 as the password.

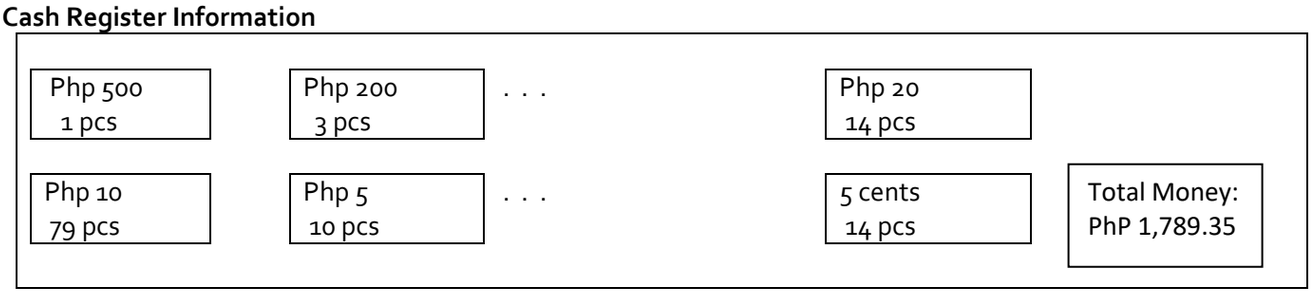
Interface Design

Design is part of the project. However, the following screens are samples to show you the required information in your interfaces.

Item Information



Cash Register Information



Also keep in mind that error checking should be done by the program. Thus, feedback (i.e., error messages) should be shown for incorrect inputs or invalid processes (e.g., cannot proceed to vend items if there are no more rice or egg).

How to Approach the Machine Project

Step 1: Problem analysis and algorithm formulation

Read the MP Specifications again! Identify clearly what are the required information from the user, what kind of processes are needed, and what will be the output (s) of your program. Clarify with your professor any issues that you might have regarding the machine project.

When you have all the necessary information, identify the necessary functions that you will need to modularize the project. Identify the required data of these functions and what kind of data they will return to the caller. Write your algorithm for each of these modules/functions as well as the algorithm for your main program.

Step 2: Implementation

In this step, you are to translate your algorithm into proper C statements. While implementing, you are to perform the other phases of program planning and design (discussed in the other steps below) together with this step.

Follow the coding standard indicated in the course notes (Modules section in AnimoSpace).

You may choose to type your program in a text editor or an IDE (i.e. Dev-C IDE) at this point. Note that you are expected to use statements taught in class. You can explore other libraries and functions in C as long as you can clearly explain how these work. You may also use arrays, should these be applicable and you are able to properly justify and explain your implementation using these. For topics not covered, it is left to the student to read ahead, research, and explore by himself.

Note though that you are NOT ALLOWED to do the following:

- to declare and use global variables (i.e., variables declared outside any function),
- to use goto statements (i.e., to jump from code segments to code segments), [gotoxy() is allowed]
- to use the break or continue statement to exit a block. Break statement can only be used to break away from the switch block,
- to use the return statement or exit statement to prematurely terminate a loop or function or program,
- to use the exit statement to prematurely terminate a loop or to terminate the function or program, and
- to call the main() function to repeat the process instead of using loops.

It is best that you perform your coding "incrementally." This means:

- Dividing the program specification into subproblems, and solving each problem separately according to your algorithm;
- Code the solutions to the subproblems one at a time. Once you're done coding the solution for one subproblem, apply testing and debugging.

Documentation

While coding, you have to include internal documentation in your programs. You are expected to have the following:

- File comments or Introductory comments
- Function comments
- In-line comments

Introductory comments are found at the very beginning of your program before the preprocessor directives. Follow the format shown below. Note that items in between < > should be replaced with the proper information.

```
/*
     Description:      <Describe what this program does briefly>
     Programmed by:    <your name here>   <section>
     Last modified:    <date when last revision was made>
     Version:          <version number>
    [Acknowledgements: <list of sites or borrowed libraries and sources>]
*/
<Preprocessor directives>

<function implementation>

int main()
{
   return 0;
}
```

Function comments precede the function header. These are used to describe what the function does and the intentions of each parameter and what is being returned, if any. If applicable, include pre-conditions as well. Pre-conditions refer to the assumed state of the parameters. Follow the format below when writing function comments:

```
/*    <Description of function>
      Precondition: <precondition / assumption>
      @param <name> <purpose>
      @return <description of returned result>
*/
<return type>
<function name> ( <parameter list>)
      :
```

Example:

```
/* This function computes for the area of a triangle
   Precondition: base and height are non-negative values
   @param base is the base measurement of the triangle in cm
   @param height is the height measurement of the triangle in cm
```

```
   @return the resulting area of the triangle
*/
float
getAreaTri (float base,
            float height)
{
    ...
}
```

In-Line Comments are ot**her comments in major parts of the code. These are expected to explain the purpose or algorithm of groups of related code, esp. for long functions.**

Step 3: Testing and Debugging

Submit the list of test cases you have used. For each feature of your program, you have to fully test it before moving to the next feature. Sample questions that you should ask yourself are:

1. What should be displayed on the screen if the user inputs an order?
2. What would happen if I input incorrect inputs? (e.g., values not within the range)
3. Is my program displaying the correct output?
4. Is my program following the correct sequence of events (correct program flow)?
5. Is my program terminating (ending/exiting) correctly? Does it exit when I press the command to `quit`? Does it exit when the program's goal has been met? Is there an infinite loop?
7. and others…

Important Points to Remember:

1. You are required to implement the project using the C language (C99 and NOT C++). Make sure you know how to compile and run in both the IDE (DEV-C++) and the command prompt (via
   `gcc -Wall -std=c99 <yourMP.c> -o <yourExe.exe>`
2. The implementation will require you to:
   - Create and Use Functions
     Note: Non-use of self-defined functions will merit a grade of **0** for the **machine project**. Too few self-defined functions may merit deductions. A general rule is to create a separate function for each option described above, unless some features are too similar that one function can serve the purpose for two [or more] of the options. Note that functions whose tasks are only to display are not included in the count for creating user-defined functions.
   - Appropriately use conditional statements, loops and other constructs discussed in class (Do not use brute force solution. **You are not allowed to use goto label statements, exit statements. You are required to pass parameters to functions and not allowed to declare global or static variables.**) Refer to Step 2 on Implementation for other details and restrictions.
   - Consistently employ coding conventions
   - Include internal documentation (i.e., comments)
3. Deadline for the project is **7:59AM of November 25, 2024 (Monday)** via submission through **AnimoSpace**. Late submission up to max of 3 days will incur deductions. That is, submissions from 8:00AM of November 25 to 7:59AM of November 26 will incur 10% deduction, submissions from 8:00AM of November 26 to 7:59AM of November 27 will incur additional 20% deduction (total of 30% deduction), submissions from 8:00AM of November 27 to 7:59AM of November 28 will incur additional additional 30% deduction (total of 60% deduction in MP score), and submissions from 8:00AM of November 28 will not be accepted anymore as facility is locked. Once locked, no MP will be accepted anymore and this will result to a **0.0** for your machine project.
4. The following are the deliverables:

Checklist:

- ❒ Upload in AnimoSpace by clicking **Submit Assignment** on Machine Project and adding the following files:
  - ❒ source code*
  - ❒ test script**
- ❒ email the softcopies of everything as attachments to **YOUR own email address** on or before the deadline

Legend:

***Source Code also includes the internal documentation. The **first few lines of the source code** should have the following declaration (in comment) **BEFORE** the introductory comment:

```
/*********************************************************************
This is to certify that this project is my own work, based on my personal efforts in studying and applying the concepts
learned. I have constructed the functions and their respective algorithms and corresponding code by myself. The
program was run, tested, and debugged by my own efforts. I further certify that I have not copied in part or whole
or otherwise plagiarized the work of other students and/or persons.
                                        <your full name>, DLSU ID# <number>
*********************************************************************/
```

****Test Script should be in a table format, with header as shown below. There should be **at least 3 distinct test classes** (as indicated in the description) **per function**. There is no need to create test scripts for functions that only perform displaying on screen (like menu where there are no conditions being checked; but for the ASCII art which is displayed depending on size and on add-ons, there are conditions

checked, thus should be included in the test script).

Function Name	#	Test Description	Sample Input (either from the user or to the function)	Expected Result	Actual Result	P/F
getAreaTri	1	**base and height measurements are less than 1.**	**base = 0.25 height = 0.75**	**…**	**…**	
	2	:::				
	3					

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the function getAreaTri(), the following are 3 distinct classes of tests:

i.) testing with base and height values smaller than 1
ii.) testing with whole number values for base and height
iii.) testing with floating point number values for base and height, larger than 1.

The following test descriptions are incorrectly formed:

Too specific: testing with base containing 0.25 and height containing 0.75
Too general: testing if function can generate correct area of triangle
Not necessary -- since already defined in pre-condition: testing with base or height containing negative values

5. **MP Demo:** You will demonstrate your project on a specified schedule during the last weeks of classes. Being unable to show up on time during the demo or being unable to answer convincingly the questions during the demo will merit a grade of **0.0** for the **MP**. The project is initially evaluated via black box testing (i.e., based on output of running program). Thus, if the program does not compile successfully using gcc -Wall -std=c99 and execute in the command prompt, a grade of 0 for the project will be incurred. However, a fully working project does not ensure a perfect grade, as the implementation (i.e., correctness and compliance in code) is still checked.
6. Any requirement not fully implemented and instruction not followed will merit deductions.
7. This is an individual project. Working in collaboration, asking other people's help, and/or copying other people's work are considered as cheating. Cheating is punishable by a grade of **0.0** for CCPROG1 course, aside from which, a cheating case may be filed with the Discipline Office.
8. The above description of the program is the basic requirement. A maximum of 10 points will be given as bonus. Use of colors may not necessarily incur bonus points. Sample additional features could be:
    - Creating a Graphical User Interface to look like a Vending Machine, use of other C libraries is allowed
    - Proper use of arrays and/or structures
    - Saving and loading the updated price and inventory count to file.

    Note that any additional feature not stated here may be added but **should not conflict with whatever instruction was given in the project specifications**. Bonus points are given upon the discretion of the teacher, based on the difficulty and applicability of the feature to the program. Note that **bonus points can only be credited if all the basic requirements are fully met** (i.e., complete and no bugs).

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honesty policy applies. Please take note that you are NOT allowed to borrow and/or copy-and-paste – in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by others including that from AI). **You should develop your own codes from scratch by yourself.**

There is only 1 final submission in the MP, but you are encouraged to upload/submit your running program in AnimoSpace to show progress in your work and to also serve as backup and for code versioning. The following Milestones are recommended:

Milestone 1 Target: October 18, 2024(F)

1.) Outline of functions, including comments, to be created with their draft descriptions in the source code. [This can still change in Phase 2 and Phase 3 but needs to be indicated already in Phase 1 to show understanding of purpose of functions.]
2.) Implementation for the functions that deal with the all the features under Maintenance Features. But, no loops will be expected yet. Thus, after setting/modifying a value, make sure to show that the value has been changed (by displaying all info about the items or all info about the cash register, whichever is appropriate) prior to exiting the program. This means that the only expectation in Phase 1 is after 1 update, the program exits.
    For the Cash Out feature, if the user chooses given an amount to cash out, just determine what quantity of each denomination is needed for the given amount to cash out. Do not check first if there are enough pieces of the required denominations.
3.) Documentation (program comments and test scripts) only for the functions that are implemented.

Milestone 2 Target: November 15, 2024 (F)

1.) Screen Flow of the entire program.
2.) Updated Phase 1 requirements by incorporating the use of loops in the options and in checking the numeric password upon entry to the Vending Maintenance Features and for choosing to Shut Down.
3.) Updated Cash Out feature based on requirement indicated in specs.
4.) Documentation (function specs and test scripts) only for the functions that are implemented, inclusive of

those that were created or updated for Phase 1.

MP Deadline: November 25, 2024 (M) 7:59AM
Entire Program as stated in specifications.