Department of Software Technology

# CCPROG1 Machine Specifications

**Deadline:** January 24, 2022 (Monday) 7:30AM via AnimoSpace

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 [Stewart-Smith, 2012]. In 2018, Japan also rolled out the pizza vending machine in Hiroshima [Roll, 2018].

In an effort to invoke appreciation of the many fascinating automation wonders of our modern era, you will re-create the internal processing of a pizza vending machine. It will handle complete functionalities from start to end of the buying process. The project expects that you design and implement the interaction to mimic vending machine responses and actions in a text-based menu format.

The following are the sequence of interaction:

1. Allow the user to choose type of crust, whether (1) thin crust or (2) thick crust.
2. Allow user to input size: 's' or 'S' for small (PhP100), 'm' or 'M' for medium (PhP175), 'l' or 'L' (PhP225) for large. FYI: Base price of pizza is based on size and already includes tomato sauce with mozzarella cheese.
3. Allow the user to choose toppings. The user should choose at least 1 topping. The user can choose as many toppings as he wants, until he chooses 'N' or 'n' referring to the next step. The user can choose the same toppings more than once. For example, if the user chooses 'h' or 'H' (to represent ham) three times, it means he wants 3 portions of ham to be placed in the pizza.
   Toppings available and their corresponding prices (for small-sized pizza) are as follows:

| Input Code | Toppings | Price |
|---|---|---|
| 'h' or 'H' | ham | 10.00 |
| 'p' or 'P' | pineapple | 6.00 |
| 's' or 'S' | sausage | 15.00 |
| 'c' or 'C' | cheese | 12.00 |
| 'o' or 'O' | olives | 10.00 |

   For medium-sized pizza, topping price indicated above is multiplied by 1.5. For large-sized pizza, topping price indicated above is multiplied by 1.75.
4. After the previous step (of choosing toppings), if the user chose cheese at least once for his topping, he is also asked the type of cheese he wants (for each cheese topping): 'M' or 'm' for mozzarella, 'G' or 'g' for gorgonzola, 'F' or 'f' for fontina, 'P' or 'p' for parmesan. That is, if the user chose cheese 3 times, he is asked to choose 3 times. He can choose the same cheese more than once if he wishes.
5. If the user chose thick crust pizza earlier, he is now given an option to choose if he wants to stuff the crust with cheese ('c' or 'C'), spam ('s' or 'S') or both ('b', 'B'). Cheese stuffing is for additional PhP30, spam stuffing is for additional PhP25, and costs additional PhP35 for both, regardless of size of pizza.
   If the user chooses 'n' or 'N', this means no stuff crust and proceed to the next step. Note that this step should not be presented to the user who chose thin crust. This step is skipped if none of the toppings chosen in the previous step was cheese.
6. The user is then asked to "insert" his payment. Note that this physical payment is replaced in your program by asking the user to press the keys associated to the bills and coins of his payment. That is: (1) 1000, (2) 500, (3) 100, (4) 50, (5) 20, (6) 10, (7) 5, (6) 1, (8) 0.25, (9) 0.10, (10) 0.05, (11) 0.01. So, if the user's pizza costs PhP360, he can choose to pay by pressing (2) only [for indicating paying PhP500] or by pressing (3) 3 times, and (4) 2 times. The user can choose 'X' or 'x' to cancel the order. If complete payment is given, the machine "dispenses" the change, if applicable. The change should also be computed and given based on the smallest number of bills and coins that can be given. For example, it should indicate breakdown of change to be given, like message of "Dispensing change: 1 – PhP100 2 – PhP20" (for the PhP500 inserted and PhP360 pizza cost). A receipt is also printed listing the transaction number, breakdown of the items

chosen, total payment, and the change. The transaction number starts at 1 (at the beginning of [each] run of the program). For every successful order, the transaction number is incremented (that is, canceled orders are not included in the count). Once complete payment is given, order cannot be canceled. Note that you are free to design how the receipt looks like, even include a name for your Pizza Vending Machine. But, items chosen should be left aligned, while currencies are 2 decimal places and aligned at the right. For example:

```
Thin Crust (M)          175.00
Ham                      15.00
Pineapple                 9.00
Mozzarella               18.00
Total                   217.00
```

7. The machine starts making the pizza while the machine counts down the time in minutes:seconds to when the pizza will be done. After which, the pizza is "dispensed" and the machine will be ready for the next customer. Note that the machine takes 2 seconds to top each chosen topping (if cheese was chosen 3 times, this is counted as 3 toppings), takes 5 seconds to do each stuffing (so if user chose 'b' for both stuffed crust, it takes 10 seconds), and takes 3 minutes to bake the pizza. There is no need to use a delay or sleep, just continuously display the countdown. For example, if based on chosen options it should take 3 minutes and 4 seconds to prepare the pizza, the excerpt of the countdown is as follows: `3:04 3:03 3:02 3:01 3:00 2:59 2:58` [and so on , of course, replaced with actual values …. until] `0:00`.

Note that there must be a way for the program to terminate (naturally) – not by closing the dialog box. Let's say that in the first step (on asking type of crust), if the input is 80808, this is a signal that the program will terminate (e.g., vending machine will be shut down for maintenance). On the other hand, for each step starting from steps 2 to 6, if the user pressed 'X' or 'x', it means he is canceling the order, so the machine needs to start from step 1 again (presumably for the next order or next customer). If wrong inputs are given in any step, the machine should produce an error message and stay at that step until an acceptable input is given.

Remember that you are to use proper conditional expressions, iterative constructs, and create subroutines (functions) to implement the requirement. You are not allowed to use exit() statement or goto statements. Check other requirements and restrictions below.

**Assumptions:**

1. The vending machine will always have enough of each ingredient (dough, topping, stuffing).
2. The vending machine will always have enough of each type of bill and coin to give as change.

There are some things not explained in the specs, like what information should be displayed. This is intentional. You can draw on (meaning, recall) interface and interaction you may have encountered when using vending machines before. Also, you put yourself in the shoes of the user and answer [at least] the following questions:

- What do I want to see on the screen when ordering?
- Do I want to see the running total of my order?
- Do I want to see quantity 0 on items I did not order or should the item not be listed at all in the receipt?
- Is the interface I designed confusing, cluttered, or missing information?

**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 Linux Kernel coding standard:
https://developer.gnome.org/documentation/guidelines/programming/coding-style.html

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),
- to use the break statement to exit a block other than switch blocks,
- 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 other 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 <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**.
   - 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.**)
   - Consistently employ coding conventions
   - Include internal documentation (i.e., comments)
3. Deadline for the project is the **7:30AM of January 24, 2022 (Monday)** via submission through **AnimoSpace**. After this time, submission facility is locked and thus 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 test functions which are only for screen design.

| 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<br>height = 0.75** | **…** | **…** | |
| | 2 | ::: | | | | |
| | 3 | | | | | |

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 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. Bonus points: You can earn up to at most 10 points bonus for additional features that your program may have. Sample features and implementation that may allow you to gain bonus points include: (1) properly and substantially using arrays and array of structure in the implementation of the project, (2) program having save and load of order information to files, (3) generating a report of how much of each ingredient (dough, topping, stuffing, bills and coins) are used, including total number of pizzas and total amount received by the machine in the run. Some of the indicated additional features may require self-study. Also, 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).

References:

Stewart-Smith, Hannah. 2012. "Japan's new 'disaster aid' vending machine". Available in: https://www.zdnet.com/article/japans-new-disaster-aid-vending-machine-video/

Roll, Dale. 2018. "The first ever pizza-dispensing vending machine in Japan is now operating in Hiroshima". Available in: https://soranews24.com/2018/08/11/the-first-ever-pizza-dispensing-vending-machine-in-japan-is-now-operating-in-hiroshima/

**HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS**

**Honesty policy applies.** Please take note that you are NOT allowed to borrow and/orcopy-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 other people that are not online). **You should develop your own codes from scratch by yourself.**