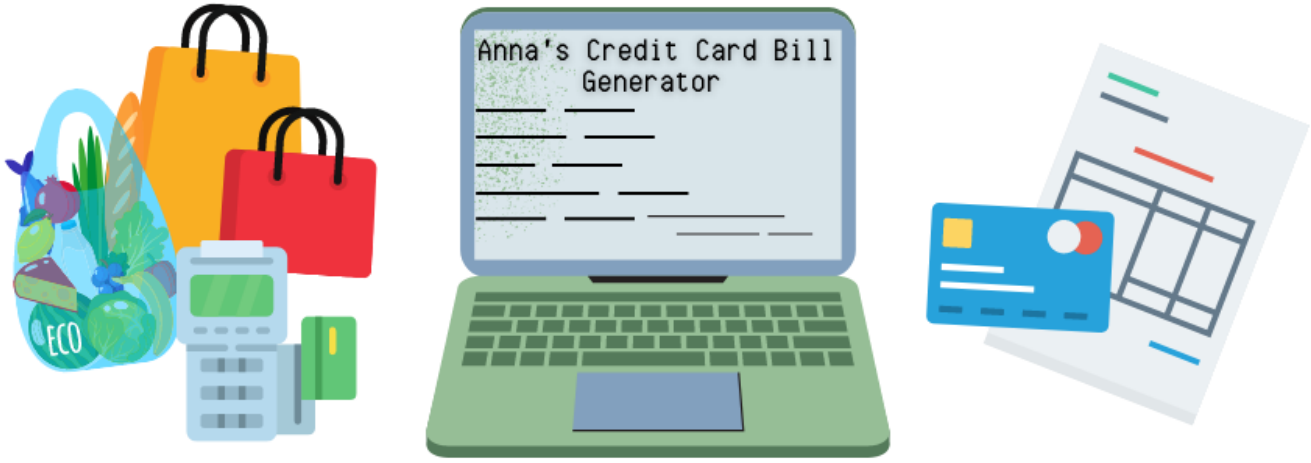
CCPROG1 Machine Specifications Final Due Date: July 31, 2023



Anna's credit card application to CCP Credit Card Bank just got approved. Anna is very careful when it comes to spending her money. She wants a way to check if her credit card bill is correct. She asked you to develop an application that can generate a credit card bill, so she can compare it with the bill given by the bank.

The application should allow Anna to input purchases, make payments, use points, view balance, view points, and generate the credit card bill. A credit card bill is generated at the end of every billing cycle. Then, the customer needs to pay for the full amount due during the next billing cycle. Otherwise, corresponding charges will apply. If there is a minimum amount due and no payment is made at all for 4 consecutive billing cycles, Anna should be notified that her credit card is about to get canceled by the bank.

The project will not require a graphical user interface and will make the most out of a text-based user interface -- something we should all be familiar with as we've worked with C.

This project is to be done **individually**.

Program Start

When the user first uses your program, they should be asked to input the credit limit. The program then proceeds to billing cycle one (1). When the application first starts, the user must start with the following values:

- Billing Cycle = 1
- Outstanding Balance = 0.00

- Previous Balance = 0.00
- (-) Payments / Credits = 0.00
- (+) Purchases = 0.00
- (+) Finance Charges = 0.00
- (+) Late Payment Charges = 0.00
- Total Amount Due = 0.00
- Minimum Amount Due = 0.00
- ETC Credit Card Bank -- Rewards Points
    - Previous Cards Points Balance = 0
    - (+) Current Points Earned = 0
    - (-) Points Used = 0
    - Total Credit Points = 0

Afterwards, transactions that can be done in the program will be displayed and the program will ask for the user action.

Game Creation

Here are all possible actions the user can do:

1. **Add purchase**
    a. The program will ask for the amount of the purchase.
    b. Transaction should be declined if the amount of purchase is greater than or equal to twice of the credit limit.
    c. If the amount is not negative,
        ■ the amount will be added to the current billing cycle's purchases
        ■ the rewards points will be calculated and added to the Current Points Earned. 1 point is earned for every 30 pesos spent. For example, if the purchase amount is 100.00, rewards points earned will be 3 points.
    d. Otherwise, the user will not be able to add the amount, an appropriate message must be displayed, and the user will be prompted for the amount again.

2. **View previous statement**
    a. The program will display the Previous Balance and the Previous Minimum Amount Due. If the current billing cycle is one, then these values are still zero.

3. **Make payment**
    a. The program will ask for the amount of the payment.
    b. If the amount is not negative, the amount will be added to the current billing cycle's payments.

c. Otherwise, the user will not be able to make the payment, a message must be displayed, and the user will be prompted for the amount again.

4. View rewards points

a. The program will display the Total Rewards Points. If the current billing cycle is one, then these values are still zero.

5. Use rewards points

a. If the total rewards points is at least 1000, the program will display the rewards that can be claimed and will ask for the user's choice. There are 3 options:
   - Php 100 eGift voucher for 1000 pts - 1000 pts will be added to the current billing cycle's used points and a message saying that the eGift voucher code was sent to the client's registered mobile number shall be displayed.
   - Php 100 credits for 1000 pts - 1000 pts will be added to the current billing cycle's used points, 100.00 will be added to Payments/Credits, and an appropriate message will be displayed to inform the user of the successful claim.
   - Cancel - the program will go back to the previous menu.
   - If the user inputs something invalid, a message must be displayed, and the user will be prompted for the amount again.

b. Otherwise, the user will not be able to claim rewards, a message saying that there are not enough points to claim anything must be displayed.

6. End billing cycle

a. When the user ends the billing cycle, a credit card bill is generated. Then, the customer needs to pay for the amount due during the next billing cycle.

b. The program will display the following as shown in the Sample Screen Output below.
   - **Previous Balance** - the total amount due of the previous billing cycle.
   - **(-) Payments / Credits** - the payments made during the current billing cycle. If it is the first billing cycle, then the previous balance is zero.
   - **(+) Purchases** - total amount of purchases made during the current billing cycle.
   - **(+) Finance Charges** - total of overlimit fee and unpaid balance fee. After calculation of these fees, the total is also added to the total amount due.
     - An overlimit fee of 500.00 is added to the finance charges when the outstanding balance is greater than the credit limit.
     - There is also a charge if the payments made during the current billing cycle is below the previous balance that must be settled. The charge is calculated by multiplying the unpaid balance by 3%. It is then added to the finance charges. For example, if the previous balance is 10000.00 and the payments made during the current billing cycle is equal to 8000.00, then the unpaid balance is 2000.00. The charge will be 60.00.

- **(+) Late Payment Charges** - charge for not paying at least the minimum amount due of the previous billing cycle. The charge is 850.00 or equivalent to the value of the unpaid minimum amount due, whichever is lower. After the calculation, this is also added to the total amount due. For example,
  - If the previous balance is 1000.00, the minimum amount due is 850.00 and the payments made during the current billing cycle is none, the late payment charge is going to be 850.00.
  - If the previous balance is 1000.00, the minimum amount due is 850.00 and the payments made during the current billing cycle is 550.00, the late payment charge is going to be 300.00.
- **Total Amount Due** - total of the outstanding balance, finance charges, and late charges. The outstanding balance is the total of the previous balance and the purchases during the current billing cycle minus the payments made during the current billing cycle. If it is the first billing cycle, then the previous balance is zero.
- **Minimum Amount Due** - a portion of the total amount due and is the minimum amount that should be paid within the next billing cycle.
  - If the total amount due is less than or equal to 850.00, the minimum amount due shall be equal to the total amount due. For example, if the total amount due is 100.00, then the minimum amount due is also 100.00.
  - If the total amount due is more than 850.00, the minimum amount due is calculated by multiplying the total amount due by 3.57%. If the minimum amount due is less than 850.00, it will be 850.00. For example, if the total amount due is 1000.00, then the minimum amount due is 850.00 because total amount due times 3.57% is less than 850.00.
- **Previous Cards Points Balance** - total previous points.
- **(+) Current Points Earned** - points earned during the current billing cycle.
- **(-) Points Used** - points used during the current billing cycle.
- **Total Credit Points** - total of previous and current points minus the used points.

c. After displaying the credit card bill, the next billing cycle will start. But before it starts, the following must be done <u>in sequence</u>:
   - The previous balance will be equal to the total amount due.
   - The previous rewards points will be equal to the total credit points.
   - The current points, used points, purchases, and payments are set to 0.
   - If there is a minimum amount due and no payment is made at all for 4 consecutive billing cycles, the credit card will be canceled. No transactions can be made with the application anymore. A message of cancellation must be displayed. The total amount due shall be displayed as the demandable amount. A penalty of equivalent to 25% of the total amount due shall also be displayed. Then, the program will end.

- 2000.00 is automatically added to the purchases of the next billing cycle every twelve (12) billing cycles.
- The billing cycle number is incremented.

d. The next billing cycle will then start.

7. Exit

a. The program will end.

If the user selected options 1 - 6, the program should continue to ask for the next transactions after the action is performed. The user must also be asked for another input again if the input is invalid -- for example, the number 35 was inputted, which doesn't correspond to anything. An error prompt should be shown to the user assuming there's an error.

Program End

The program will end

- when the credit card is canceled; or,
- when the user selects **"Exit"**. Thus, do not forget to implement an exit.

How to Approach the Machine Project

Step 1: Problem analysis and algorithm formulation

Read the MP Specifications again! Identify clearly 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),
- 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.

Note that creation of user-defined functions are necessary for this project. 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 2: 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**. 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.**)

- Consistently employ coding conventions
- Include internal documentation (i.e., comments)

3. Deadline for the project is the **7:30AM of July 31, 2023 (Monday)** via submission through **AnimoSpace**. After this time, the 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 create test scripts for functions that only perform displaying on screen. See the sample below.

Function Name	#	Test Description	Sample Input (either from the user or to the function)	Expected Result	Actual Result	Pass /Fail
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 the questions convincingly 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 the running program). Thus, if the program does not compile successfully using gcc -Wall 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 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 a bonus. Use of colors may not necessarily incur bonus points. Sample additional features could be:
   (a) use of arrays or other data structures substantially, appropriately, and properly. This requires you to research the appropriate constructs. NOTE: Just because arrays or data structs are advanced topics does not mean that it is appropriate to be used all the time.

   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 other people that are not online). **You should develop your own codes from scratch by yourself.**