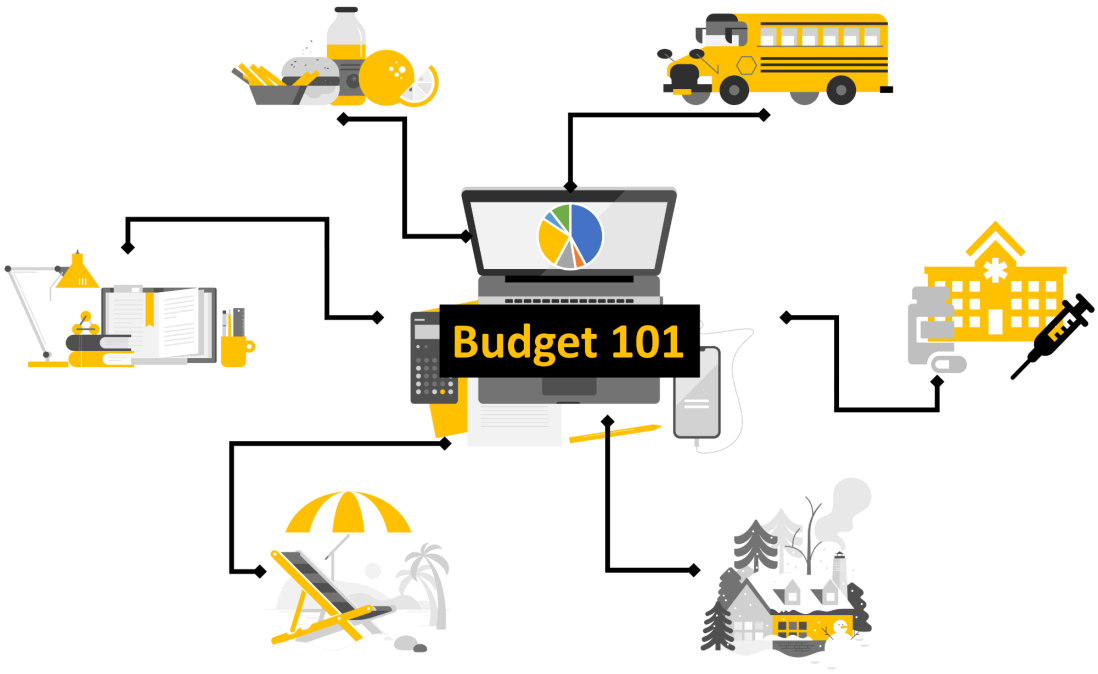
CCPROG1 Machine Specifications Final Due Date: 5 December, 2022



Anna is an incoming student in DLSU. She lives quite far from DLSU, so she has to rent a space which will cost her around Php 4,000.00. To make sure that she at least eats everyday, her parents already arranged a meal plan subscription for her which will cost her Php 300.00 a day. A fixed monthly allowance is given to her and nothing more. Anna has to learn how to budget her allowance. Help her by creating a budgeting application.

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**.

Begin Budgeting

When the user first uses your program, they should be asked for two (2) inputs: their monthly allowance, and their budget allocation.

- **Monthly allowance** - basis of the monthly budget. This amount will be added to the current balance every month. The minimum amount needs to be Php 13,000.00. If the user inputs a lower amount, a prompt must be displayed, and the user will be asked for the amount again.
- **Budget Allocation** - budget that the user will allot for each of the categories of expenses, and for savings every month. There are 6 categories:

(a) **food and drink**, (b) **transportation**, (c) **healthcare**, (d) **housing and utilities**, (e) **leisure**, and (f) **school.** Please note that if the amounts do not equate to the current balance, your program must prompt the user for the budget allocation again. Also, the minimum amount for the house and utilities must be Php 4,000.00, and Php 300.00 * 30 days = Php 9,000.00 for the food and drink; if not, your program must prompt the user for the budget allocation again.

Afterwards, the program proceeds to Day One (1). When the first day starts, the user must start with the following values:

- Current day = 1
- Expenses for each category = 0.00
- Savings = 0.00
- Balance = equivalent to their monthly allowance

Start of the Day

At the start of each day, the following must be performed **in order**:

1. If it is the start of a new month,
    a. Add the monthly allowance to the balance.
    b. Reset both the current month's expenses and savings to 0.00.
    c. Ask for the new budget allocation for the month.
    d. Deduct Php 4,000.00 from the balance, then add it to the current month's expenses for Housing and Utilities.
2. Deduct Php 300.00 from the balance, then add it to the current month's expenses for Food and Drink.
3. The day counter, total savings, balance, current month's expenses for each category, and current month's savings must be displayed.
4. The user is presented a list of actions they can select from, or they can choose to exit the program.

User Actions

Here are all possible actions the user can do:

A. Add expenses
    - The user will be asked to select a category. There are 6 categories:
      (a) **food and drink**, (b) **transportation**, (c) **healthcare**, (d) **housing and utilities**, (e) **leisure**, and (f) **school**

- After selecting a category, the user will be asked for the amount they are about to spend.
- If the amount is lower than or equal to the balance, the current month's expense for the category will be updated. Otherwise, the user will not be able to add the amount, a message must be displayed, and the user will be prompted for the amount again.
- NOTE: The current month's expense for each category resets to 0.00 every month (i.e. after every 30 days from Day 1).

B. Add savings
   - The user will be asked for the amount they want to save.
   - If the amount is lower than or equal to the balance, the total savings and the current month's savings will be updated. Otherwise, the user will not be able to add the amount, a message must be displayed, and the user will be prompted for the amount again.
   - NOTE: The current month's savings resets to 0.00 every month (i.e. after every 30 days from Day 1).

C. Use savings
   - The user will be asked for the amount they want to transfer from their total savings to their balance.
   - If the amount is lower than or equal to the total savings, the amount must be transferred from the total savings to the balance. Otherwise, the user will not be able to transfer the amount, a message must be displayed, and the user will be prompted for the amount again.
   - NOTE: The total savings must NEVER reset to 0.00 every month.

D. End the day, or month
   - The user has the option to end the current day, or month.
   - Months are counted with respect to Day 1. A month is counted as 30 days, so the start of a new month is always every after 30 days from Day 1 (i.e. Days 1, 31, 61, 91, 121, …).
   - When ending the day, the player will no longer be able to select any actions for the current day.

If the user selected actions A, B, or C, the program should continue to ask for the next actions after the action is performed. The user must also be asked for another input again if the input is invalid -- for example the letter H was inputted, which doesn't correspond to anything. An error prompt should be shown to the user assuming there's an error.

Also, please note that the player does not have to perform actions A to C and may directly choose action D to end the day or month.

End of the Day

After choosing action D which is end the day or month, the following must be performed **in order**:

1. If the user chose to **end the month**,
    a. For every day that has passed, deduct Php 300.00 for the meal plan subscription, then add it to the current month's expenses for Food and Drink (e.g. If the current day is 18, the end of the current month will be 30. 12 days would have passed, so your program must automatically deduct 12 days * Php 300.00 = Php 3,600.00 from the balance, then add it to the current month's expenses for Food and Drink)
    b. Display how many days have passed.
    c. Update the day counter to the end of the month. (e.g. If the current day is 18, the day counter will be 30.)
2. The day counter, balance, and total savings must be displayed.
3. If it is the **end of the month**, budget reports will be displayed. The reports must show whether the expenses for each category is within the budget allocated or not, and whether the user saved more than what's committed or not.
4. Increment the day counter by 1.
5. The next day will start.

Program End Condition

The program will end

- If the user selects **"Exit"**, so do not forget to implement an exit.
- OR, if the user runs out of money before the end of the month.

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…

SUGGESTED MILESTONES: [INCLUDES CODING, DOCUMENTATION, AND TESTING]

Here is a suggested set of tasks to guide you in completing all requirements. Note that you are not expected to submit milestone requirements. All requirements in this specs are due on December 5, 2022 8am.

Milestone 1: Target due October 21, 2022

1. Display of introduction (if you want).
2. Function for getting the monthly allowance.
3. Function for getting the budget allocation.
4. Function/s for the start of the day.
5. Function/s for user actions menu.

Milestone 2: Target due November 18, 2022

1. Function/s for adding expenses.
2. Function/s for adding savings.
3. Function/s for using savings.
4. Function/s for ending the day.
5. Update the program to be able to start the next day.
6. Implement exit function.

Milestone 3: Target due December 2, 2022

1. Apply error checking when getting the monthly allowance.
2. Apply error checking when getting the budget allocation.
3. Apply error checking when adding expenses, adding savings, and using savings.
4. Apply error checking when asking for choices.
5. Function/s for budget reports.
6. Function/s for ending the month.

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 **8:00AM of December 5, 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 create test scripts for functions that only perform displaying on screen.

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) ask for the name of the user which would be used by the program to make responses that are more centered on the user (i.e. customized feedback).
   (b) use of data structures substantially, appropriately, and properly. This requires you to research the appropriate constructs.

   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).


DEPARTMENT OF
SOFTWARE TECHNOLOGY

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.**