De La Salle University
College of Computer Studies
Department of Software Technology

CCPROG1 Machine Project Term 1 AY2024-2025
Energon Cubes Production

Background:

In Transformers, the inhabitants called Cybertronians are powered by Energon. It is their power source. Soundwave is a character who can create Energon Cubes for his team. Energon Cubes are what they consume in order for them to continue functioning. It is already after the war for Cybertron and all is at peace but there is still a need for Energon Cubes. Soundwave has decided to help his fellow Cybertronians and produce the cubes in exchange for Energon. He has a **goal** to **earn 1,000,000 Energon** in order for him to comfortably survive and continue to produce Energon Cubes.

Soundwave starts with an initial 10,000 Energon.

Mechanics:

- **Generating Energon Cubes** – At the start of each week(Sunday), Soundwave produces Energon Cubes. The amount of Energon required to produce cubes is different each start of the week since Soundwave is already way past his prime(getting old). The amount of Energon can cost as little as 80 to a maximum of 120 Energon to be able to generate 1 Energon Cube. **He is only able to generate cubes in stacks of 10** so if the cost is 90 Energon, generating 1 stack will cost 900 Energon. A confirmation will be displayed after entering the amount of stacks to be produced. If they choose no(n), then they will be asked again how many stacks would they want to produce.

  NOTE: He does not have the ability to generate a single cube. It must always be per stack (10 cubes)!

```
Week 1 Day 1
Energon Storage: 10000  Stacks:0

Production cost for this week is 89 Energon for 1 cube.
It will cost 890 Energon to produce 1 stack.

How many stacks do you wish to produce for this week? 11
11 stacks will cost 9790 Energon, proceed?(y/n) y
11 stacks produced.
```

- **Expiring Energon Cubes** – The generated cubes **expire at the end of the 7th day (Saturday)**. Meaning if you do not sell it before the end of Saturday, it will have gone to waste.

- **Earning Energon** – A nearby store owned by Swindle, buys Energon Cubes in exchange for Energon. Throughout the week, his buying price changes daily depending on the Energon Surge Trend.

  NOTE: He buys only by the stack! He does not buy single cubes!

```
Week 1 Day 2
Energon Storage: 210    Stacks: 11

Swindle is buying Energon Cubes for 101 Energon per cube.
You can earn 1010 Energon per stack.

How many stacks do you wish to sell to Swindle? 6
6 stacks are about to be sold, proceed?(y/n) y
6 stacks sold.
You earned 6060 Energon.

Week 1 Day 3
Energon Storage: 6270   Stacks: 5

Swindle is buying Energon Cubes for 87 Energon per cube.
You can earn 870 Energon per stack.

How many stacks do you wish to sell to Swindle? 5
5 stacks are about to be sold, proceed?(y/n) y
5 stacks sold.
You earned 4350 Energon.
```

- **The Energon Surge Trends are as follows:**
    - **Scrap trend** – Prices for the week of this trend will have a minimum of 20 energon and a maximum price equivalent to the generation cost -10 at the start of the week.

      Example: If the cost to generate an Energon Cube was 90 Energon, the max price this trend can produce this week is 80. This means that you can only earn 800(80 x 10) Energon for a single stack.

    - **Nominal trend** – Prices for the week of this trend will have a minimum of 80 Energon and a maximum price equivalent to 105% of the generation cost at the start of the week.

      Example: If the cost to generate an Energon Cube was 90 Energon, the max price this trend can produce this week is 94. This means that you can earn 940(94 x 10) Energon for a single stack.

    - **Primus trend** – Prices for the week of this trend will have a minimum price equivalent to the generation cost at the start of the week and a maximum price of 400% of the generation cost of the start of the week.

      Example: If the cost to generate an Energon Cube was 90 Energon, the max price this trend can produce this week is 360. This means that you can earn 3600(360 x 10) Energon for a single stack.

  NOTE: Energon Surge Trends are random and not announced! This means that you can only know what the trend is by observing the how much Swindle will be offering to pay each day!

- **End Goal** – Soundwave is targeting to **make 1 million Energon in 10 weeks**! The game will end after the 10th week whether or not he achieves this goal or not. A message congratulating him will be displayed if he makes the goal otherwise display a consoling message if he doesn't before the program ends and terminates. The game will also end if you do not have enough Energon to produce 1 stack(10 pcs) of Energon Cubes!

  NOTE: The game will continue even if Soundwave manages to make 1 million Energon before the end of the 10th week. The game should NOT end before the 10th week just because he already has 1 million Energon.

- **BONUS: Recycle Old Energon Cubes (Optional)** – Instead of Energon Cubes expiring at the end of the week, they end up in the recycle pile. Recycled Energon Cubes can still be sold but only for 1/3 of the price that Swindle offers.

It is expected that you will make use of all the following concepts in this project:

- User-defined functions
- Pointers
- Loops
- Conditions
  For random number generation, you may refer to https://www.tutorialspoint.com/rand-and-srand-in-c on how to use rand() and srand().

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.

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.

Please include a dev feature that will allow the user to modify the amount of Bells and also modify the current week. This will make it easier to do testing during the demo. This feature should be accessible after the program starts.

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 continue,
- to call return 0 in the middle main function (return 0 should only be at the end of the main function), 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;
- coding 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. Items in between ***[ ]*** are optional, indicate if applicable.

```
/*
    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 after a user input?

2. What would happen if inputs are incorrect? (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?
6. 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.

MP Deadline: November 25, 2024 (M) 7:59AM

Entire Program as stated in specifications.