Department of Software Technology

# CCPROG1 Machine Specifications

**Deadline:** December 5, 2022 (Monday) 8:00AM via AnimoSpace

Numbers are universal. In whatever country you are in, 2 + 2 is always equal to 4. In this project, you are to create a program that will deal with number sequences. A number sequence is a series of (in this case) integers that are linked by a rule or pattern. There are three options in the Main Menu of your program. These are (1) Guessing the pattern to generate the fifth number in the sequence; (2) Generating n numbers in a series given the pattern to follow; and (3) Exit. [You can shorten the names of the options in the menu]

**The first option** (in the menu) indicated above is typically presented as part of an IQ (Intelligence Quotient) exam, where the person guesses the 5th number in the series. In this case, it is the user who gives the four (4) numbers in a series, and the program will deduce which among the following patterns it falls under (for all, if any, that applies) and produces also the 5th number in the series:

- Increments of N
- Decrements of N
- Even Series
- Odd Series
- Prime Series
- +M +N Series
- -M -N Series
- +M -N Series
- -M +N Series
- *N Series
- Binary Series

The following are examples that should be considered in your program.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Case 1: | Given: | 11 | 23 | 35 | 47 | | |
| | | Answer: 59 | | | | Reason/s: | Increments of 12 |
| Case 2: | Given: | 1 | 3 | 5 | 7 | | |
| | | Answer: 9 | | | | Reason/s: | Increments of 2<br>Odd series |
| Case 3: | Given: | 12 | 14 | 16 | 18 | | |
| | | Answer: 20 | | | | Reason/s: | Increments of 2<br>Even series |
| Case 4: | Given: | -3 | 0 | 3 | 6 | | |
| | | Answer: | | 9 | | Reason/s: | Increments of 3 |
| Case 5: | Given: | 10 | 8 | 6 | 4 | | |
| | | Answer: | | 2 | | Reason/s: | Decrements of 2<br>Even series |
| Case 6: | Given: | 12 | 13 | 13 | 14 | | |
| | | Answer: | | 14 | | Reason/s: | +1 +0 sequence |
| Case 7: | Given: | 5 | 7 | 11 | 13 | | |
| | | Answer: | | 17 | | Reason/s: | Prime series<br>+2 +4 sequence |
| Case 8: | Given: | 33 | 50 | 12 | 29 | | |
| | | Answer: | | ? | | Reason/s: | No relation! |
| Case 9: | Given: | 12 | 29 | 33 | 50 | | |
| | | Answer: | | 54 | | Reason/s: | +17 +4 sequence |
| Case 10: | Given: | 50 | 33 | 29 | 12 | | |
| | | Answer: | | 8 | | Reason/s: | -17 –4 sequence |
| Case 11: | Given: | 7 | 10 | 17 | 27 | | |
| | | Answer: | | ? | | Reason/s: | No relation! |
| Case 12: | Given: | 5 | 15 | 45 | 135 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Answer: | | 405 | | Reason/s: | *3 sequence |
| Case 13: | Given: | 5 | -15 | 45 | -135 | | |
| | | Answer: | | 405 | | Reason/s: | *-3 sequence |
| Case 14: | Given: | 3 | 2 | 4 | 3 | | |
| | | Answer: | | 5 | | Reason/s: | -1 +2 sequence |
| Case 15: | Given: | 11 | 100 | 101 | 110 | | |
| | | Answer: | | 111 | | Reason/s: | Binary series |

**Note:**

1. For the reason/s, the numbers should change depending on the input sequence (i.e., Increments of *N*, where *N* can vary).
2. In Increments and Decrements, the N should be a positive value only.
3. Those patterns involving M and N, the M and N signify numbers that can vary and the numbers cannot be the same.
4. For the prime, odd, even, and binary series, the values should be sequential in the series, otherwise, the result should be No Relation.
5. Do not assume that the initial number used to compute is 1.
6. The input numbers can be negative. The input numbers can also be more than 3 digits. You may assume that the number can be stored into an integer variable.
7. Any other cases of sequence numbers not discussed in these specifications are not included (exceptions are those indicated for bonus features. Therefore, the program should not output the 5th integer and the reason should be "No Relation" or some other similar notation.
8. Apart from the possibility of having multiple reasons for the same Resulting $5^{th}$ number, it is also possible to have different Resulting $5^{th}$ numbers and the corresponding reasons per set of inputs.
9. The program should keep on asking for input sequences until the first input number (in a sequence) is the sentinel *-999*. When the sentinel is given, the program reverts to the Main Menu .

**Upon entering the second option** (in the Main Menu), the program first asks the user to choose which among the following will be generated:

1. Increments of N
2. Decrements of N
3. Incrementing Even Series
4. Incrementing Odd Series
5. Incrementing Prime Series
6. **[Go back to Main Menu]**

If the user chooses either option 1 or 2 (increments or decrements), the user is asked to input value of N. There should be error checking done by the program that the given N should be a positive number before proceeding to the next step. If it is not a positive number, an error message is displayed and the program reverts back to the beginning of the second option menu choices (the choices 1 – 6 above).

The user is also asked to input a value X, representing the number of values to be generated. This X should be at least 2. [There is no limit to this value, but assume the number and the computations/results can be stored in integer variable/s.] If X is not at least 2, the program displays an error message and reverts back to the beginning of the second option menu choices (the choices 1 – 6 above).

When inputs above is/are valid, the user is tasked to input another value Y, representing the first number in the series. Should this Y not be a valid first number given the choice (e.g., for choices 3 - 5 above), an error message is displayed and the program reverts back to the beginning of the second option. Refer to examples A to C below:

**Example A:** Let's say user chooses Incrementing Odd Series, entered 8 for X and entered 5 for Y. The program should display the following before going back to the menu options: 5 7 9 11 13 15 17 19

**Example B:** Let's say user chooses Incrementing Prime Series, entered 3 for X and entered 4 for Y. The program should display the following before going back to the menu options: Starting number given is not prime.

**Example C:** Let's say user chooses Decrements of N, entered 4 for N, entered 5 for X and entered 2 for Y. The program should display the following before going back to the menu options: 2 -2 -6 -10 -14

When all inputs are valid, the results are displayed on the screen. The program then reverts back to the beginning of the second option.

When user chooses option 6, the program reverts to the Main Menu. When the user chooses option 3 in the Main Menu, the program terminates properly. Note that calling exit and calling main() are NOT allowed.
[Continue reading below for other reminders, requirements, and restrictions.]

## 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),
- 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 o**ther 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…

### **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 requirement. All requirements in this specs is due on December 5, 2022 8am.

**Milestone 1**: Target due October 21, 2022

1. Functions for the menu options.
2. Function for getting input 4 integers
3. Function/s to check [and compute next] if increments and decrements
4. Function/s to check [and compute next] if odd or even series
5. Function to check [and compute next] if *N series

**Milestone 2**: Target due November 18, 2022

1. Function to check [and compute next] if +M +N series
2. Function to check [and compute next] if +M -N series
3. Function to check [and compute next] if -M -N series
4. Function to check [and compute next] if -M +N series
5. Function to check if prime number
6. Function/s to generate increments/decrements series
7. Function/s to generate odd/even series

**Milestone 3**: Target due December 2, 2022

1. Function to convert binary to decimal
2. Function to convert decimal to binary
3. Function to generate next prime number/s
4. Update program for proper menu transitions.

**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 | P/F |
|---|---|---|---|---|---|---|
| getAreaTri | 1 | **base and height measurements are less than 1.** | **base = 0.25<br>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 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:
   (a) other sequences, like Fibonacci series, Automorphic numbers, Ugly numbers, and Narcissistic numbers. This requires you to research on the concept.
   (b) use of arrays or dynamic lists. This may require you to read ahead or research on how to apply these properly to your project.

   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.**