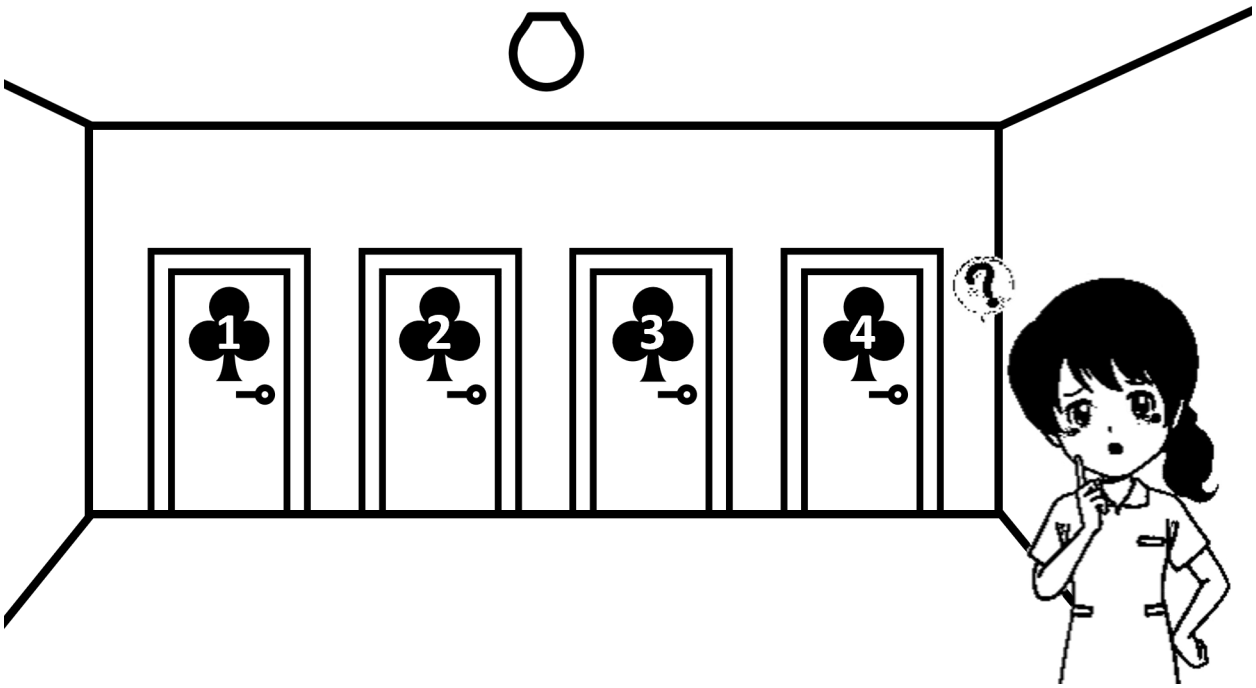
CCPROG1 Machine Specifications Final Due Date: April 12, 2023



Four of Clubs or "Good Fortune, Bad Fortune" is a game of chance. There will be 4 doors for each room. One leads to the next room; one leads to an empty room; one leads to a room of chance; and, one leads to death. The user is initially given 3 lives. Every death reduces the number of lives. The user wins if they reach the exit. The user loses if they run out of lives. Create a C program of the game.

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 program starts, the user will start with the following values:

- Lives = 3
- Number of Rooms = 5
- Correct Doors = 32341
- Death Doors = 44123
- Chance Doors = 13412
- Room Number = 1

The main menu of the game should have three (3) options: **Game Creation**, **Game Play**, and **Exit**.

Game Creation

When the user chooses to create a game, the user should be asked for four (4) inputs in sequence: number of rooms, correct doors, death doors, and chance doors.

- **Number of Rooms** - basis for the number of digits of each of the other inputs: correct, death, and chance doors. The value can be from 3 to 8. If the input is valid, the number of rooms must be updated. If the user inputs a value that is out of range, an error message must be displayed, then the user will be prompted to input the value again.

- **Correct Doors** - the right path to the exit. The value should be an n-digit number, where n corresponds to the number of rooms. Each digit indicates which door for each room leads to the next safe room; thus, the digit is either 1, 2, 3, or 4. The sequence of the correct doors in the value is from left to right. If the input is valid, the value must be updated. If the user inputs a value with an incorrect number of digits or invalid digit, an error message must be displayed, then the user will be prompted to input the value again.

- **Death Doors** - the door that decreases the number of lives. The value should be an n-digit number, where n corresponds to the number of rooms. Each digit indicates which door leads to death; thus, the digit is either 1, 2, 3, or 4. The sequence of the death doors in the value is from left to right. The value must not overlap with the correct doors. If the input is valid, the value must be updated. If the user inputs a value that overlaps with the correct doors, or a value with an incorrect number of digits or an invalid digit, an error message must be displayed, then the user will be prompted to input the value again.

- **Chance Doors** - the door that has a chance of death, empty room, or showing the correct door number. The value should be an n-digit number, where n corresponds to the number of rooms. Each digit indicates which door leads to a chance room; thus, the digit is either 1, 2, 3, or 4. The sequence of the chance doors in the value is from left to right. The value must not overlap with the correct doors, and death doors. If the input is valid, the value must be updated. If the user inputs a value that overlaps with the correct and/or death doors, or a value with an incorrect number of digits or an invalid digit, an error message must be displayed, then the user will be prompted to input the value again.

Afterwards, the program goes back to the main menu.

Game Play

When the user chooses to play the game, the game will start following the inputs of the user during the Game Creation or the initial values stated in the Program Start section of this specification. These should not be shown to the player. The goal of the player is to guess the correct doors that lead to the safe rooms until the exit is found. The exit door is the last digit of Correct Doors. The program should run following the steps below.

1. The current room number, and number of lives must be displayed.
2. Four (4) doors will be shown to the player. Only the door numbers are shown to them. They should not be given any hints or information about whatever lies on the other side of the doors.
3. The players will then choose which door to open. Only one door can be opened at a time.
4. Using the values of Correct Doors, Death Doors, and Chance Doors, the program will check what kind of room the door leads to.
    - If it leads to the **Correct Room**, the player will proceed to the next safe room where the next set of doors will be shown.
    - If it leads to the **Death Room**, the number of lives of the player will be reduced by 1.
    - If it leads to the **Chance Room**, the program must randomly choose whether
        - The player gains a life (10% chance)
        - The player will lose a life (20% chance)
        - The chance room will just be an empty room (40% chance)
        - The door number that leads to the empty room will be shown to the player (10% chance)
        - The door number that leads to the death room will be shown to the player (10% chance)
        - The door number that leads to the correct room will be shown to the player (10% chance)
    - If it leads to the **Empty Room**, nothing will happen.

    For example,

    Given the following values: Lives = 1, Correct Doors = 14232, Death Doors = 42123, Chance Doors = 23414, Room Number = 3

    If the player chooses Door #1, it will lead to a Death Room since the current room number is 3 and the third death door is Door #1. The player will lose a life. Since the player lost all lives, the player loses and the game is over.

5. The program must display a message regarding the result of the choice made by the player. Regardless of the door opened, the player should have the option to go back to the previous rooms and open another door, unless the player runs out of lives or reaches the exit door.

Program End

The game play will be over when

- Losing condition: The player runs out of lives.
- Winning condition: The player reaches the exit door.

The program will end 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 **8:00AM of April 12, 2023 (Wednesday)** 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:

CCPROG1 Machine Specifications Final Due Date: April 12, 2023

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).

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS


DEPARTMENT OF
SOFTWARE TECHNOLOGY

CCPROG1 Machine Specifications Final Due Date: April 12, 2023

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.**