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

CCPROG1 Machine Project

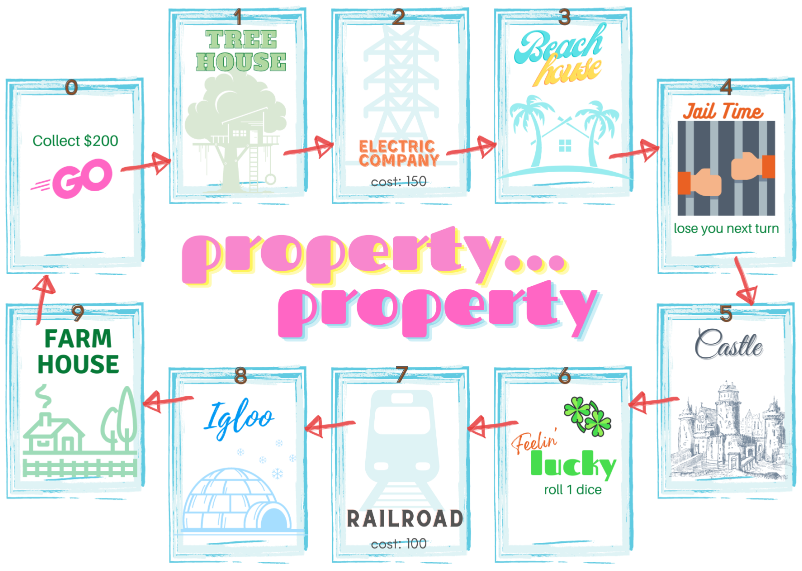
Property...Property

Submit on or before **Jan 24, 2022 (M) 0730**

For this project, you are to create a software version of a game called **Property...Property**. This is a turn-based two-player board game. Players compete to acquire wealth by buying or renting properties. The game ends when a player goes bankrupt, i.e. he does not have enough money to pay rent.

The board

The players take turn in moving around the game board shown below. The board has 10 spaces (position 0 to position 9). Note that your implementation is text-based, i.e. no images will be displayed on the screen.



Setting up for the game

At the start of the game, players do not own any property. All properties on the board are owned by the Bank. All properties owned by the Bank are up for sale and are not renovated. The Bank has unlimited cash. The smallest denomination in this game is 1.

Each player has an initial cash-on-hand of 200. They can use this to buy properties or pay rent during the game.

Initially, both players are at **Go** (position 0).

Playing the game

Players take turn in rolling the dice to move clockwise around the board. At each turn, the player rolls a dice to determine the number of spaces he moves on the board. The player may land on **Go** (position 0), on **Jail Time** (position 4), on **Feelin' Lucky** (position 6), or on a property (house properties at positions 1, 3, 5, 8, 9, electric company at position 2, and railroad at position 7). Based on the space where the player lands on, different actions may take place.

Player lands on a property.

- If the property is unowned, the player may choose to buy the property from the bank. The player pays the bank the cost of the property.
- If the property is owned by another player, the player must pay rent to the owner.
- If the property is a house property that is owned by this player, the player may choose to either do nothing, or renovate the property if it has not been renovated yet. The cost of renovation is 50.
- If the property is not a house property or is a renovated property owned by this player, the player does nothing during this round.

Player lands on Jail Time. The player loses his **next** turn.

Player lands on or passes by Go. The player collects 200 from the Bank.

Player lands on Feelin' Lucky. The player rolls a dice to determine his *luck*.

When one player goes bankrupt, the game ends and the other player is declared the winner. A player is considered bankrupt when he is unable to pay rent even after selling all his properties to the bank.

Buying properties

When a player lands on a property that is owned by the Bank, the player can choose to buy that property from the Bank by paying the property cost. The player must have enough cash-on-hand to pay for the cost, otherwise he cannot buy the property.

Electric Company	The cost of the Electric Company is 150.
Railroad	The cost of the Railroad is 100.
House Properties	The cost of the house properties is computed based on its position on the board. The cost of the house property is 20 times the remainder of the position of the property when divided by 7 *E.g. Tree House at position 1 costs 20.*

Paying rent

When a player lands on a property that is owned by another player, he has to pay the owner rent for this property.

Electric Company	The rent for landing on the Electric Company is 8 times the value of the dice rolled. *For example, it is player 1's turn and he is currently at the Farm House (position 9). Player 1 rolled the dice and the value is 3. Player 1 moves 3 spaces and lands on Electric Company (position 2) that is owned by player 2. Player 1 pays 24* ($8 \times 3 = 24$) *to Player 2, the rent for landing on the Electric Company.*
Railroad	The rent for landing on the Railroad is 35.
House Properties	The rent of the house properties is computed based on its cost. The rent of an unrenovated property is one-fifth of its cost. The rent of a renovated property is one plus twice the rent (unrenovated). *E.g. The cost of the Beach House (position 3) is 60. The rent is 12. If it has been renovated, its rent is 25.*

Not enough cash! When a player does not have enough cash-on-hand to pay for rent, he can sell his properties back to the Bank until he collects enough cash to pay for rent. The player selects a property to resell back to the Bank (one property at a time). The resell value of the property is half the cost of the property. When all the properties of a player are sold to the bank and his cash is not enough to pay the rent, this player is considered bankrupt and the game ends.

Feelin' Lucky

When a player lands on **Feelin' Lucky** (position 6), the player rolls the dice to determin his *luck*. His *luck* is determined by the dice value.

Dice Value	Action
Dice value is a prime number.	The player collect an amount from the Bank. The amount is a value in $[100, 200]$
Dice value is not a prime number.	The player pays an amount to the Bank. The amount is a value in $[50, 150]$
Dice value is 1.	The player goes to Jail, thus loses his next turn.

The amount that the player will collect from or pay to the Bank is a randomly generated value within the specified range.

Important notes on implementation

The project is considered wrong when the following are not implemented accordingly.

1. Cost, rent, rent for renovated properties **must** be computed when needed. These values must not be stored in different variables or in an array for each of the properties.
2. In keeping track of the ownership of properties, you are not allowed to use seven different variables or use an array. You must keep track of the ownership and status of the properties as a 9-digit integer.

9-digit integer	Notes
$000x0x000$	Each digit corresponds to one property on the board. • The first digit from the right is the status of Tree House (property 1). • The second digit from the right is for Electric Company (property 2), the third digit from the right is for Beach House (property 3) and so on. • The leftmost digit (9th digit from the right) is the status of the Farm House (property 9). The digits on each property are updated when a player buys, renovates or resells a property. The 4th and the 6th digits from the right can be any digit since these are **Jail Time** and **Feelin' Lucky** spaces, and are not owned by anyone.

	owns the property	renovates the property
Player 1	1	3
Player 2	2	4
Bank	0	NOT APPLICABLE

3. The program is menu-driven. When the program is executed, the user is presented with option in the form of a **menu**. The user may choose to either Start a new game, or Exit the program. When the user chooses to Start a new game, the game starts. When the game ends, the menu is presented to the user again. The program ends only when the user chooses the Exit option. If not appropriate, Continue? Yes or No? questions should not exist in the program.

Bonus features

The following are not part of the basic requirements for this project. Correct implementation of additonal features may incur a maximum of 10 bonus points. Some bonus features include, but not limited to:

1. **Configuration option.** The user may change zero or more the initial settings of the game, which includes initial cash value, rent of Railroad, multiplier for rent of Electric Company, ranges of amounts in **Feelin' Lucky**, and the cost of renovation.
2. **Game end condition.** The user may set the criteria for a game to end. The basic requirement states that game ends when a player goes bankrupt. The user may *activate* another game end condition when a specified cash-on-hand amount is reached, game ends. The user may set which of these conditions (or both) must be satisfied to end a game.

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. 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` statement to exit a block other than `switch` blocks,
- to use the `return` 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;
- 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>
    Precondiiton:  <precondition / assumption>
    @param  <name>   <purpose>
    @return  <description of returned result>
*/
<return type>  <function name>  ( <parameter list> )
{
    <implementation>
}
```

Example:

```
/*
    This function computes for the area of a triangle
    Precondition:  base and height are non-negative values
    @param base   the base measurement of the triangle in cm
    @param height   the height measurement of the triangle in cm
    @return the resulting area of the triangle
*/
float getAreaTri (float base, float height)
{
    <implementation>
}
```

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:

(a) What should be displayed on the screen after the user enters a value?
(b) What would happen if input is incorrect? (e.g. value not within the range, value entered not among choices, etc)
(c) Is my program displaying the correct output?
(d) Is my program following the correct sequence of events (correct program flow)?
(e) 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?

(f) 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**.
   - Appropriately use conditional statements, loops and other constructs discussed in class (Make sure to comply to the list of **NOT ALLOWED** listed under **Step 2** above. You are required to pass parameters to functions and not allowed to use global or static variables.)
   - Consistently employ coding conventions
   - Include internal documentation (i.e., comments)

3. Deadline for the project is the 7:30AM of January 24, 2022 (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** 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♣
  - ☐ declaration of original work♠
- ☐ email the softcopies of everything as attachments to YOUR own email on or before the deadline

Legend:

♣ Test Script should be in a table format as shown below. There should be **at least 3 distinct test classes** (as indicated in the description) **per function**. There is no need to test functions which are only for screen design.

Function: `float getAreaTri(float base, float height)` *// function name for the test cases below*
Function description: computes for the area of the triangle *//short description of what the function does*

#	Test Description	Sample Input *either from the user or passed to the function*	Expected Result	Actual Result	P/F

1.	base is below 0	`base = -1`
		`height = 1.2`
			return 0.0	. . .	. . .

2.					
3.					

Function: *// function name for the test cases below*
Function description: *//short description of what the function does*

#	Test Description	Sample Input *either from the user or passed to the function*	Expected Result	Actual Result	P/F
1.	. . .	. . .	. . .	. . .	. . .
2.					
3.					

♠ This is an **individual project** and must be designed, created, developed and completed by yourself. No part of the code submitted should come from another source (online, books, other people). Place in the comment section the following to certify that the project you submitted is your own work.

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 all 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*⟩

5. **MP Demo:** You will present your project on a specified schedule during the last weeks of classes. Being unable to show up on time during your demo schedule or being unable to answer convincingly the questions during the demo will merit a grade of **0** for your machine project. 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` 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, borrowing or copying other people?s work or from books or online sources (either in full or in part) 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. **Bonus points:** You can earn up to **at most 10 points bonus** for additional features that your program may have. Sample features and implementation that may allow you to gain bonus points include those enumerated in the **Bonus features** above. Some of the indicated additional features may require self-study.

   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).