# Project 5: Monopoly

*Due Fri 24 October at 11pm*

## Background

Monopoly(™) is an American-originated board game published by Parker Brothers, subtitled as "The Fast-Dealing Property Trading Game." Players move around the gameboard depending on the face-value of the rolled dice, buy or trade properties, develop the properties with houses and hotels, then collect rent from their opponents, with the goal to drive others into bankruptcy.

In this project, you will write a simulator for a simplified version of the game, coupled with a simple strategy. In the course of building the simulator, you will exercise the implementation of C++ classes, and will gain further experience using file I/O and arrays or vectors.

## Rules of the Game

The official Monopoly game employs a board consisting of 40 spaces: 28 properties (22 colored streets, four railway stations, and two utilities), three Chance spaces, three Community Chest spaces, a Luxury Tax space, an Income Tax space, and four corner squares—GO, (In) Jail/Just Visiting, Free Parking, and Go to Jail. Note well that the Project 5 version of Monopoly differs significantly from the official game, in the service of simplification. Your implementation must follow the rules as defined here; ignore anything you may read in the Parker Brothers manual.

### *Monopoly Board*

The monopoly board will be described in an input file, with each line specifying a location (i.e., space on the board). There will only be one "GO" space in the board and the players will start in this location. The remaining spaces are either properties, or "Parking" and no other types of spaces will appear. A line gives the name of the location, a number (price for properties, zero otherwise), and a color (or "NONE" for "GO" and "Parking"). The property's valid color inputs will be one of the following: RED, BLUE, GREEN, YELLOW, BLACK. If a player owns all the properties of a given color, we say this player has a *monopoly* on that color.

## *Movements/Game*

Each player will start with a specified cash amount as indicated in one of the input files. Instead of rolling dice to determine how many steps to move forward, we employ a deck of numbered cards for this purpose. Every round the players each take a turn in sequence, by grabbing the top card from the deck and moving the specified number of steps. For example if the player grabs a card that says “2”, that player moves two steps forward from its current location. Once all the cards are used, we restart from the beginning of the deck without any shuffling.

If a player lands on the “GO” space, the player receives $2000 from the bank.

If a player lands on a “Parking” space, nothing happens.

If a player lands on an unowned property, the player may buy the property. Whether it does so depends on its *strategy*, as described below. If the player does not buy, the move is a simple visit and the property remains unowned.

If a player lands on a property owned by another player, the player must pay rent to the owner. The basic rent is 10% of the property’s purchase price, rounded to the nearest integer dollar ($.50 gets rounded up to $1). If the owner has a monopoly on this property’s color, then rent is doubled. Each house on the property also doubles the rent. For example, if the owner has a monopoly with two houses, then the base rent is multiplied by eight.

If a player steps on a property they own, they may build a house on the property, according to its strategy. A property can support at most two houses. The cost of building a house is a fixed price of $1000.

Once a property is bought, the player retains that property until the end of the game, unless they go bankrupt. A player goes bankrupt if they owe more rent than they can pay with available cash. In that case, all the cash and property with houses of the bankrupt player gets transferred to the player owed the rent. The bankrupt player is then removed from the game. To remove a player, simply skip their turn on all subsequent rounds. (Player numbers are retained.)

The game ends when only a single player remains, or when the specified number of rounds is finished. When the game ends, your program outputs a summary of results and terminates.

If more than one player remains at the end, the winner is the one with the greatest wealth, defined as the sum of property value (assessed at purchase price, not counting houses) and cash. If more than one player ends with the same wealth, the lower-numbered tied player win.

## Strategy

There are many different strategies to play a monopoly game, some players might keep more cash on hand to encounter rent while others might buy more properties. In this project we will use two parameters to define a player's game strategy: Threshold on cash balance (a multiple of purchase price) to buy a property ($\theta_p$) and a threshold on cash balance (an absolute amount) to buy a house ($\theta_h$).

The threshold on cash balance to buy a property ($\theta_p$) is used to determine whether a player will buy the property or not. The player's cash balance must be greater or equal to the product of the threshold and the property's price for the player to purchase the property. To buy a house, the player's cash balance must be strictly greater than the threshold ($\theta_h$) for the player to make the decision to build a house.

## Inputs and File IO/ Error Checking

You are given three file-names as input, in three separate lines (without any prompt), followed by a fourth line specifying the number of rounds for the simulation.

file1: This name will be provided to you on the first line. This file specifies the locations on the board, one per line. Each line contains, in order, separated by spaces:

- A string with the location name (a property, "GO", or "Parking"). The string will have no embedded whitespace.
- A price for purchase (0 for Go or Parking), readable as integer.
- A string giving the color. For Go and Parking, the color is specified as "NONE".

file2: This name will be provided to you on the second line. This file contains the initial cash balance for each player, and the sequence of cards used to determine the number of steps to take on each move. Each line of this file will be a number, readable as an integer. Your program must check that they are all positive integers.

file3: This name will be provided to you on the third line. This file contains the player's information. Each line represents the strategy used by a player and consists of a threshold on cash balance to buy property, and a threshold on cash balance to build a house, in order, separated by a space. You can assume all the thresholds provided are valid, with $\theta_p$ as double and $\theta_h$ as integer, and there will be at least one player.

number of rounds: This is the fourth input line. A round is completed when all players have finished a move starting from player 0.

You are responsible for checking:

a) All the specified price of properties are positive
b) No other colors are included

c) The initial cash balance and cards only contain positive integer numbers

If any of these are not true, your program should print the following message on a new line, with no leading spaces, followed by a newline, and exit:

```
Improper inputs.
```

This should be to standard output, not standard error. Don't forget the period at the end!

## Output

As you simulate the game, you must print the following information. In each of these statements any element inside angle brackets <...> must be replaced with the proper value, without the brackets. Be careful to produce the proper spacing between such values and surrounding text. Each statement is to be printed on its own line, with no leading spaces, followed by a newline.

- Announce when simulation starts

  ```
  ***MONOPOLY GAME STARTS***
  ```

- Announce the beginning of each round. The first round is round 1.

  ```
  Round: <round#>
  ```

- Announce the players movement and actions, there can be these following actions
  - Move to the property and stays

    ```
    Player <player#> moves <#steps> step(s) to <Property Name>
    and stays.
    ```

  - Move to the property "GO" and earns $2000

    ```
    Player <player#> moves <#steps> step(s) to GO and receives
    $2000.
    ```

  - Move to the property and purchases the property

    ```
    Player <player#> moves <#steps> step(s) to <Property Name>
    and purchases <Property Name>.
    ```

  - Move to the property and pay rent to other player

    ```
    Player <player#> moves <#steps> step(s) to <Property Name>
    and pays $<rent> rent to Player <player#>.
    ```

- Move to the property and bankrupt, then transfer property to other player

  ```
  Player <player#> moves <#steps> step(s) to <Property Name>
  and bankrupt, transfers property to Player <player#>.
  ```

- Move to own property and build house

  ```
  Player <player#> moves <#steps> step(s) to <Property Name>
  and builds house number <#house>.
  ```

After the simulation ends, you will need to output the results and the game summary of each player in the following format.

- Announce the simulation result with the winner

  ```
  ***SIMULATION RESULTS***
  Player <player#> wins the game with total asset value of $<asset
  values>.
  ```

- Announce the game summary for each player

  ```
  ***GAME SUMMARY***
  Player <player#>:
  Cash Balance: $<cash balance>
  Number of Purchased Properties: <#num properties>
  ```

  - If the player bankrupt and is out of the game, simply output the following statement after Player <player#>:

    ```
    Bankrupt and out of game.
    ```

  - If the player does not have any property, simply output the following line after the statement Cash Balance: $<cash balance>.

    ```
    No purchased property.
    ```

# Sample Runs

*Sample 1:*
You are provided with three files: board.txt, player.txt and card.txt

board.txt: a board with 16 locations
player.txt: information about 3 players
card.txt: specifies players' initial cash and 20 different steps for cards

**The given rounds limit is 3**

```
***MONOPOLY GAME STARTS***
Round: 1
Player 0 moves 2 step(s) to AnnArbor and purchases AnnArbor.
Player 1 moves 3 step(s) to GrandRapids and purchases GrandRapids.
Player 2 moves 1 step(s) to Detroit and purchases Detroit.
Round: 2
Player 0 moves 4 step(s) to Arcadia and purchases Arcadia.
Player 1 moves 2 step(s) to Troy and purchases Troy.
Player 2 moves 5 step(s) to Arcadia and pays $250 rent to Player 0.
Round: 3
Player 0 moves 3 step(s) to GrandHaven and purchases GrandHaven.
Player 1 moves 5 step(s) to Kentwood and purchases Kentwood.
Player 2 moves 1 step(s) to Parking and stays.
***SIMULATION RESULTS***
Player 0 wins the game with total asset value of $10250.
***GAME SUMMARY***
Player 0:
Cash Balance: $1750
Number of Purchased Properties: 3
Player 1:
Cash Balance: $2400
Number of Purchased Properties: 3
Player 2:
Cash Balance: $6250
Number of Purchased Properties: 1
```

*Sample 2:*
You are provided with three files: board_small.txt, player_small.txt and card_small.txt

board_small.txt: a board with 10 locations
player_small.txt: information about 2 players
card_small.txt: specifies players' initial cash and 10 different steps for cards

**The rounds given is 5**

```
***MONOPOLY GAME STARTS***
Round: 1
Player 0 moves 1 step(s) to Los_Angeles and purchases Los_Angeles.
Player 1 moves 2 step(s) to Irvine and purchases Irvine.
Round: 2
Player 0 moves 1 step(s) to Irvine and pays $280 rent to Player 1.
Player 1 moves 4 step(s) to Parking and stays.
Round: 3
Player 0 moves 6 step(s) to San_Diego and purchases San_Diego.
Player 1 moves 5 step(s) to Los_Angeles and pays $600 rent to Player 0.
Round: 4
Player 0 moves 2 step(s) to GO and receives $2000.
Player 1 moves 5 step(s) to Parking and stays.
Round: 5
Player 0 moves 1 step(s) to Los_Angeles and builds house number 1.
Player 1 moves 5 step(s) to Los_Angeles and pays $1200 rent to Player 0.
***SIMULATION RESULTS***
Player 0 wins the game with total asset value of $12520.
***GAME SUMMARY***
Player 0:
Cash Balance: $7020
Number of Purchased Properties: 2
Player 1:
Cash Balance: $5680
Number of Purchased Properties: 1
```

# Requirements

In this project, you are required to define at least ***2 classes***: One for the properties and one for the player. Your implementation should open and read the information into appropriate data types, then close the files. You are also responsible for following the style guidelines as updated for this assignment (to be posted shortly).

# Grading

- 10 points for submitting something that compiles
- 10 points for employing proper style, as described by the updated C++ style guidelines (to be posted)
- 35 points for correctness on a variety of test inputs