# 2: One Player

CSCI 6626 /4526 Spring 2018

## 1 Goals

- To build a second simple class that will interact with the other classes in the future.
- To implement and use an enumeration type.
- To use the `extern` storage class.

## 2 The Player Class

This class represents one player. We are ready to implement some of its functions; more functions will be added later.

**Data members.** A player has

- A name like RedWolf or RubyMan (no spaces)
- A color: a constant from the color enumeration, colorEnum. (See below.)
- A integer score, the number of columns this player has captured in this game, initially 0.
- A personal scoreboard for the current game (an array of 3 integers), containing the numbers of the columns that have been captured. When the third column is captured, the player wins the game.

**Functions.**

- A constructor with two parameters: name and color.
- Appropriate destructor, print, and `operator<<` functions.
- colorEnum color()
  An accessor function. Return the player's color.
- int score()
  An accessor function. Return the number of columns that have been captured in this game.
- bool wonColumn( int colNum)
  Store the column number in the next available slot in the scoreboard and increment the counter. Return `true` if the player has won 3 columns, `false` otherwise. Eventually, this will be called by Column::stop() to register the fact that the specified column has been captured. For this stage of the program, call it from the Player-unit-test function.

## 3 Using Enumerations

To use enum types effectively, you need a civilized way to input an enum constant and to output it. Numeric codes are not civilized. The problem is that several parts of a program typically need to use the enumeration, so it cannot be put into a single class.

**Enum input.** Supply a menu of possible color choices and ask the player to input a single character as the selection. Use the char in a switch statement to select which enum variable to store in the Player object.

**Output.** To output an enum constant, you should create an array of output strings that is parallel to your enum list. This array cannot go into an .hpp file with the enum declaration since, if that .hpp file were included in more than one compile module, the array object will be compiled more than once and be part of two .o files. The linker will then be unable to link the application because it will see two objects with the same name.

The solution to this linking problem is to use the extern storage class. Identical extern declarations can be included by several files, but this does not cause a linking problem because an extern declaration does not create an object. (It instructs the linker to look for an object that was created elsewhere.) One copy of the extern declaration *with an initializer* is written in the .cpp file that matches the .hpp file containing the enum declaration. This creates the object that all other modules can and will link to.

### 3.1 Define the enumerations.

Create a file called enums.hpp

- Into it, put an enum declaration for ColorEnum, and list the five colors you need: white, orange, yellow, green, blue. Some people may wish to add a sixth constant, for errors.
- For each enum declaration, you need an array of strings for civilized output. (See "The defining declaration", below.)
- #include enums.hpp in the .hpp file for every class that needs to use the colors.
- For output, use an enum constant to subscript this array of words:
  `cout <<Color:  <<" "<<words[playerColor];`

**The defining declaration.** One cpp file must declare the array of color names, *with an initializer*. This is the "defining declaration" for the extern variable. The declarations have this general form:

```
extern const char* words[3] = {"Andy","Bob","Carla"};
```

Put the definition for this string array in `enums.cpp`. Later, more enums will be added to the list.