CSCI 3180 Assignment 3 Report

1.

For the Perl with dynamic scoping, we can implement the “PayDue” method of the Player class with the dynamic scoping feature of Perl. The dynamic scoping feature can allow us to temporarily change the value of parameters inside "PayDue" method of Player class through changing the package variables inside the package "Player". The implemetation of "payDue" method is shown with the following code.

```perl
sub payDue {
    my $self = shift;
    $self->{money} += $income * (1 - $tax_rate);
    $self->{money} -= $due * (1 + $handling_fee_rate);
}
```

And when we need to call the "payDue", we can change the parameters for the method tempoaraily with defining the "local" variable with certain values. The code examples is as follows:

```perl
local $Player::due = $price; # $price is the calculated price for the due
$main::cur_player->payDue();
```

The above code demonstrates the code for "chargeToll" inside "Land" class. We can change the due variable to value "price" and then call the "payDue" with the local variable values as the parameters for this calling. For the "local" variables that are not defined before call the "payDue", the "payDue" will use the "default value" for those variables and the "default value" is defined through the initial value of the package variable.

For the Python, we use static variable instead for similar function. The implementation of "payDue" method is similar as follows except the package variables are replaced with static variable.

```python
    def payDue(self):
        self.money += Player.income * (1 - Player.tax_rate)
        self.money -= Player.due * (1 + Player.handling_fee_rate)
```

And when we call the "payDue" method, we need to change the static variable of "Player" class in order to set the parameters for this method. The example code is as follows:

```python
Player.due = price
Player.income = 0
Player.tax_rate = 0.2
Player.handling_fee_rate = 0
cur_player.payDue()
```

Int this case, we need to set all values of the class variables in "Player" to pass the parameters. As there is no "dynamic scoping" in Python, the value of the class variables will be permanently changed and thus whenever we call the method, we need to set all the values for the class variables and do not have any "default value" for those class variables as parameters.

2.

The "local" keyword in Perl provides the option of using "dynamic scoping". While the keyword "my" in Perl provides the function to use "static scoping", "local" keyword can server as a complement for the scoping options for Perl and allows us to simplify some codes with this function.

Originally, the Perl language only allows global variables and thus there are much difficulty for the efficiency of using this language. Users need to consider other variables defined globally all through the program. Then, the keyword "local" was introduced to temporarily change the value (of use the name of the variable) and increase the programming efficiency. Specially, the "local" variable was determined at runtime and the difference calling order for the subroutines could result in different values for the "local" variable. Then another keyword "my" was also introduced in Perl to separately support the "static scoping" feature. After that, Perl managed to support both scoping types and became more flexible when implementing programs with these features.

For the application of the keyword "local", we could use this feature to redirect all the parameters of a certain name to another object and change the value of parameters whithin all the functions all at once. This feature allows us to define functions that share the same parameters while we do not need to pass the value of parameters to every function but just use "local" or the dynamic scoping to set the value for the parameters. And as the values are only temporarily changed, we do not need to restore the values and the efficiency is increased.

For my opinion, the "local" keyword in Perl provides another choice for scoping. And we can freely select the most suitable scoping type for each task. Unlike other languages that only support one kind of scoping, this keyword can be used whenever necessary. All in all, for me, more option is always better. And as "local" can provide users another option, then it is valuable.