M01 Introduction to C#

Author: Yi Chen
Date: 2020.01.15

Subject

Table of Contents

List of Tables

List of Figures

1. M01 Introduction to C#

Create C# classes to represent a company salary system. Use inheritance to merge (inherit) these classes in a sensible way, and compute.

2. Project Requirements

Class:

- *Abstract person class (firstname, lastname, address, salary, performance rating, method computePayCheck, method computePayRaise)*
- *An employee is a person*
- *Average Employee class*
- *Programmer Employee class.*
- *Manager Employee class.*

Find interesting ways to compute paychecks and pay rises.

Proper OOP is expected:

- *Use proper inheritance*
- *Use proper scope of variables*

Main method:

- *Create one object of each (Average , Programmer, and Manager)*
- *Populate the fields (select whatever values you wish)*
- *Place objects into an ArrayList (https://docs.microsoft.com/en-us/dotnet/api/system.collections.arraylist?view=net-5.0)*
- *Make a loop to compute pay raise and write output to console.*
- *Use the performance rating property to calculate increase.*
- *Make a loop to compute paycheck.*
- *Assume deductions such as taxes and pension.*
- *Write to console the paycheck summary in tabular format (information ordered in columns).*
- *Add a ReadLine() statement or some other method at the end of your Main method to prevent the console app from terminating.*

2.1. Derived Requirements

- *A prepared PDF document with your commented code.*
- *Document should have relevant meta information and be well formatted.*
- *Console app files (executable, libraries, json) in zipped format (do not submit a .exe file to Brightspace).*

3. Create C# classes to represent a company salary system

Paystub calculation system.

3.1. Abstract person class

Abstract person class (firstname, lastname, address, salary, performance rating, method computePayCheck, method computePayRaise)

Programmer.cs | AverageEmployee.cs | Employee.cs | Person.cs | Program.cs | Manager.cs

ConsoleApp1 | ConsoleApp1.Person

```
using System;
using System.Collections.Generic;
using System.Text;


namespace ConsoleApp1
{
    public abstract class Person // base class (parent)
    {
        public string Firstname;
        public string Lastname;
        public string Address;
        public decimal Salary;
        public decimal benefit;
        public decimal PerformanceRating;


        public abstract decimal ComputePayCheck(decimal cpp_value, decimal fed_tax, decimal ns_tax);

        public abstract decimal ComputePayRaise(decimal salary, decimal performanceRating);

    }


}
```

3.1.1. An employee is a person

```
Programmer.cs   AverageEmployee.cs   Employee.cs   Person.cs   Program.cs   Manager.cs
ConsoleApp1   ConsoleApp1.Employee   ComputePayRaise(decir
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;

namespace ConsoleApp1
{
    class Employee : Person   // derived class (child)
    {
        //Calculate the basic pay-period exemption = basic yearly exemption (3500) / PAY_PERIOD
        private const decimal BASIC_PAY_PERIOD_EXEMPTION = 3500;/* The basic yearly exemption ($3,500 for 2021) by the number */
        private const decimal PAY_PERIOD = 26;
        private const decimal EMPLOYEE_CONTRIBUTION = 0.0545m;
        public const decimal EI = 889.54m;  /*The maximum EI premiums in 2021 :889.54m*/


        /*Calculate the CPP which employee have to deduct from the Bi-weekly gross salary in 26 times a year*/
        public decimal CPP(decimal salary, decimal benefit_c)
        {
            Salary = salary;
            benefit = benefit_c;
            decimal cpp_annual;
            decimal cpp_contributions;
            decimal deducted_basic_pay_period;
            decimal pensionable_income;
            pensionable_income = Salary + benefit;
            deducted_basic_pay_period = pensionable_income - BASIC_PAY_PERIOD_EXEMPTION;
            cpp_contributions = deducted_basic_pay_period * EMPLOYEE_CONTRIBUTION;
            if (cpp_contributions > BASIC_PAY_PERIOD_EXEMPTION)
            {
                cpp_annual = BASIC_PAY_PERIOD_EXEMPTION;
                return cpp_annual;
            }
            else
            {
                cpp_annual = cpp_contributions;
                return cpp_annual;
            }
```

3.1.2. Average/ Programmer/ Manager Employee class

Average Employee inherited Employee.

```
Programmer.cs   AverageEmployee.cs   Employee.cs   Person.cs   Program.cs   Manager.cs
ConsoleApp1   ConsoleApp1.AverageEmployee   AverageEmployee(string firs
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp1
{
    class AverageEmployee : Employee //derived from Employee (child of Employee)
    {

        public AverageEmployee(string firstName, string lastname, string address, decimal salary, decimal performanceRating)
        {
            Firstname = firstName;
            Lastname = lastname;
            Address = address;
            Salary = salary;
            PerformanceRating = performanceRating;

        }


    }
}
```

Programming inherited Employee.

Subject

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp1
{
    class Programmer : Employee //derived from Employee (child of Employee)
    {
        public Programmer(string firstName, string lastname, string address, decimal salary, decimal performanceRating)
        {
            Firstname = firstName;
            Lastname = lastname;
            Address = address;
            Salary = salary;
            PerformanceRating = performanceRating;

        }

    }
}
```

Manager inherited Employee

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp1
{
    class Manager : Employee //derived from Employee (child of Employee)
    {
        public Manager(string firstName, string lastname, string address, decimal salary, decimal performanceRating)
        {
            Firstname = firstName;
            Lastname = lastname;
            Address = address;
            Salary = salary;
            PerformanceRating = performanceRating;

        }
    }

}
```

3.1.3. Part 2: Use inheritance to merge (inherit) these classes in a sensible way, and compute:

Table 1: compute paychecks and pay rises

Subject

```
        /*Calculate paystub*/
        public override decimal ComputePayCheck(decimal cpp_value, decimal fed_tax, decimal ns_tax)
        {
            decimal payCheck;
            decimal cpp_ = cpp_value;
            decimal federal_tax = fed_tax;
            decimal NS_tax = ns_tax;
            payCheck = Salary - federal_tax - NS_tax - cpp_ - EI;

            return payCheck;

        }

        /*Calculate pay rise*/
        public override decimal ComputePayRaise(decimal salary, decimal performanceRating)
        {
            Salary = salary;
            Performance_Rating = performanceRating;
            decimal rised_salary = Salary * Performance_Rating;
            return rised_salary;
        }
    }
```

Tax , CPP and EI

Subject

mer.cs | AverageEmployee.cs | Employee.cs | Person.cs | Program.cs | Manager.cs

leApp1 | ConsoleApp1.Employee

```
        }

        /*Calculate Fedral Tax */
        public decimal FedTax(decimal taxble_income)
        {
            var fed_tax_rate = new List<decimal> { 0.15m, 0.205m, 0.26m, 0.29m, 0.33m };
            decimal getTaxRate;
            decimal federal_tax;
            decimal income = taxble_income;


            foreach (decimal taxRate in fed_tax_rate)
            {
                if (taxble_income < 49020)
                {
                    getTaxRate = fed_tax_rate[0];
                    /*Console.Write($"{getTaxRate} ");*/
                    federal_tax = (income - 0) * getTaxRate + 0;
                    return federal_tax;
                }
                else if (taxble_income > 49020 && taxble_income <= 98040)
                {
                    getTaxRate = fed_tax_rate[1];
                    /*Console.Write($"{getTaxRate} ");*/
                    federal_tax = (income - 49020) * getTaxRate + 7353;
                    return federal_tax;
                }
                else if (taxble_income > 98040 && taxble_income <= 151978)
                {
                    getTaxRate = fed_tax_rate[2];
                    /*Console.Write($"{getTaxRate} ");*/
                    federal_tax = (income - 98040) * getTaxRate + 17402.10m;
                    return federal_tax;
                }
                else if (taxble_income > 151978 && taxble_income <= 216511)
                {
                    getTaxRate = fed_tax_rate[3];
                    /*Console.Write($"{getTaxRate} ");*/
                    federal_tax = (income - 151978) * getTaxRate + 31425.98m;
                    return federal_tax;
                }
                else
                {
                    getTaxRate = fed_tax_rate[4];
                    /*Console.Write($"{getTaxRate} ");*/
                    federal_tax = (income - 216511) * getTaxRate + 50140.55m;
                    return federal_tax;
                }
            }

            return 0;
```

NS Tax

Subject

```cpp
/*Calculate Nova Scotia Tax */
public decimal NSTax(decimal ns_taxble_income)
{
    var ns_tax_rate = new List<decimal> { 0.0879m, 0.1495m, 0.1667m, 0.175m, 0.21m };
    decimal getNSTaxRate;
    decimal ns_tax;

    foreach (decimal taxRate in ns_tax_rate)
    {
        if (ns_taxble_income <= 29590)
        {
            getNSTaxRate = ns_tax_rate[0];
            /*Console.Write($"{getNSTaxRate} ");*/
            ns_tax = (ns_taxble_income - 0) * getNSTaxRate + 0;
            return ns_tax;
        }
        else if (ns_taxble_income > 29590 && ns_taxble_income <= 59180)
        {
            getNSTaxRate = ns_tax_rate[1];
            /*Console.Write($"{getNSTaxRate} ");*/
            ns_tax = (ns_taxble_income - 29590) * getNSTaxRate + 2601;
            return ns_tax;
        }
        else if (ns_taxble_income > 59180 && ns_taxble_income <= 93000)
        {
            getNSTaxRate = ns_tax_rate[2];
            /*Console.Write($"{getNSTaxRate} ");*/
            ns_tax = (ns_taxble_income - 59180) * getNSTaxRate + 7025;
            return ns_tax;
        }
        else if (ns_taxble_income > 93000 && ns_taxble_income <= 150000)
        {
            getNSTaxRate = ns_tax_rate[3];
            /*Console.Write($"{getNSTaxRate} ");*/
            ns_tax = (ns_taxble_income - 93000) * getNSTaxRate + 12662;
            return ns_tax;
        }
        else
        {
            getNSTaxRate = ns_tax_rate[4];
            /*Console.Write($"{getNSTaxRate} ");*/
            ns_tax = (ns_taxble_income - 150000) * getNSTaxRate + 22637;
            return ns_tax;
        }

    }

    return 0;
}
```

Cpp

Subject

```
/*Calculate the CPP which employee have to deduct from the Bi-weekly gross salary in 26 times a year*/
public decimal CPP(decimal salary, decimal benefit_c)
{
    Salary = salary;
    benefit = benefit_c;
    decimal cpp_annual;
    decimal cpp_contributions;
    decimal deducted_basic_pay_period;
    decimal pensionable_income;
    pensionable_income = Salary + benefit;
    deducted_basic_pay_period = pensionable_income - BASIC_PAY_PERIOD_EXEMPTION;
    cpp_contributions = deducted_basic_pay_period * EMPLOYEE_CONTRIBUTION;
    if (cpp_contributions > BASIC_PAY_PERIOD_EXEMPTION)
    {
        cpp_annual = BASIC_PAY_PERIOD_EXEMPTION;
        return cpp_annual;
    }
    else
    {
        cpp_annual = cpp_contributions;
        return cpp_annual;
    }
    return 0;
}
```

EI

```
public const decimal EI = 889.54m;  /*The maximum EI premiums in 2021 :889.54m*/
```

3.2. Main method

Figure 1: Main method

- Create one object of each (Average , Programmer, and Manager)

```
namespace ConsoleApp1
{
    abstract class Program
    {
        public static void Main(string[] args)
        {

            var employee = new Employee();
            var average_Employee = new AverageEmployee("Yi", "Chen", "Dartmouth", 44000, 0.3m);
            var programmer = new Programmer("Anna", "Sage", "Dartmouth", 83000, 0.9m);
            var manager = new Manager("Amily", "Zhang", "Dartmouth", 150000, 0.5m);
```

- Populate the fields

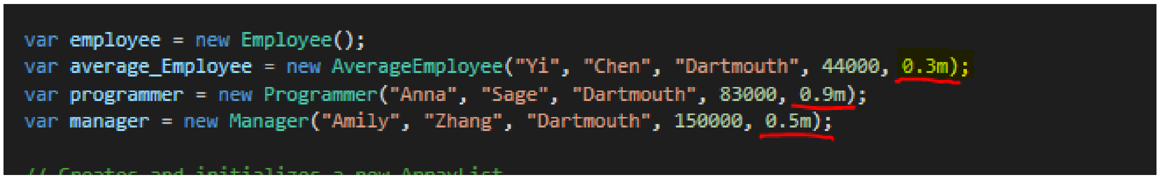
```
    /*Calculate the CPP which employee have to deduct from the Bi-weekly gross salary in 26 times a year*/
    public decimal CPP(decimal salary, decimal benefit_c)
    {
        Salary = salary;
        benefit = benefit_c;
        decimal cpp_annual;
        decimal cpp_contributions;
        decimal deducted_basic_pay_period;
        decimal pensionable_income;
        pensionable_income = Salary + benefit;
        deducted_basic_pay_period = pensionable_income - BASIC_PAY_PERIOD_EXEMPTION;
        cpp_contributions = deducted_basic_pay_period * EMPLOYEE_CONTRIBUTION;
```

- Place objects into an ArrayList (https://docs.microsoft.com/en-us/dotnet/api/system.collections.arraylist?view=net-5.0)
- Make a loop to compute pay raise and write output to console.

```
    // Creates and initializes a new ArrayList.
    ArrayList myAL = new ArrayList();
    myAL.Add(average_Employee.Salary);
    myAL.Add(programmer.Salary);
    myAL.Add(manager.Salary);

    ArrayList myARL = new ArrayList();
    myARL.Add(average_Employee.PerformanceRating);
    myARL.Add(programmer.PerformanceRating);
    myARL.Add(manager.PerformanceRating);
/*  Console.WriteLine($"|AverageEmployee First name | {average_Employee.Firstname}      | Lastname | {average_Employee.Lastname}    | the address | {average_Employee.Address} | the salary is: {avera
    Console.WriteLine($"|Programmer First name      | {programmer.Firstname}    | Lastname | {programmer.Lastname}    | the address | {programmer.Address} | the salary is: {programmer.Salary} |");
    Console.WriteLine($"|manager First name         | {manager.Firstname}   | Lastname | {manager.Lastname}   | the address | {manager.Address} | the salary is: {manager.Salary} |");*/
    foreach (decimal grossySalary in myAL)
    {
        foreach (decimal riseRate in myARL)
        {
            var paystub = employee.ComputePayCheck(employee.CPP(grossySalary, 600), employee.FedTax(grossySalary), employee.NSTax(grossySalary));
            var payRising = employee.ComputePayRaise(grossySalary, riseRate);
            Console.Write("| paystub     |   "  + paystub + "    | pay rising       |   " + payRising);
            /* Console.Write("      " + grossySalary + "      " +   paystub + "      "   +  riseRate    +"     " + payRising);*/
            /*Console.Write($"| grossy pay  | {grossySalary}       |    paystub     | { paystub}      | riseRate    | {riseRate} |  pay rised  |  {payRising} |" );*/
            /* Console.Write($"| Grossy salary    | {grossySalary}   | paystub is      | {paystub}       |");*/
            /* Console.WriteLine($"| Grossy salary    | {grossySalary}   | wage increased  | {payRising}          |");*/
        }  ≤ 2ms elapsed
        /* Console.WriteLine();*/
    }
    Console.ReadLine();
}
```

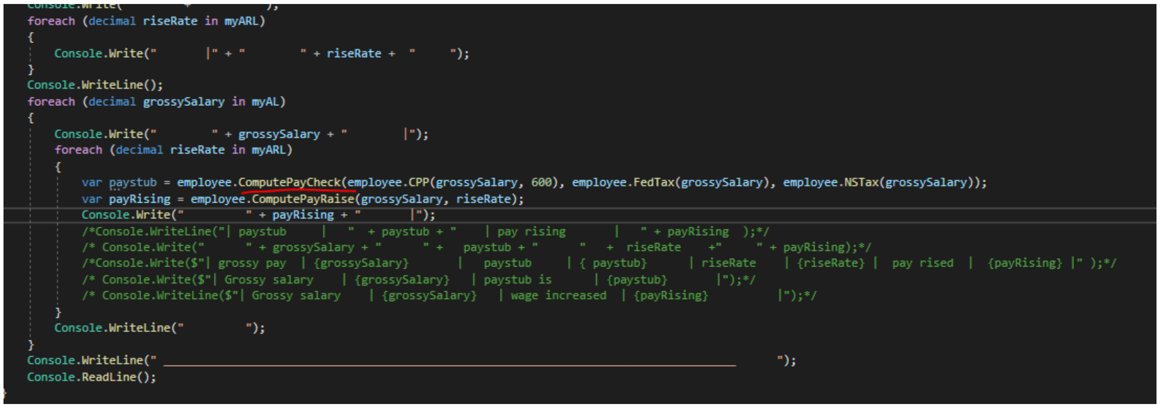
- Use the performance rating property to calculate increase.

```
ArrayList myARL = new ArrayList();
myARL.Add(average_Employee.PerformanceRating);
myARL.Add(programmer.PerformanceRating);
myARL.Add(manager.PerformanceRating);
```



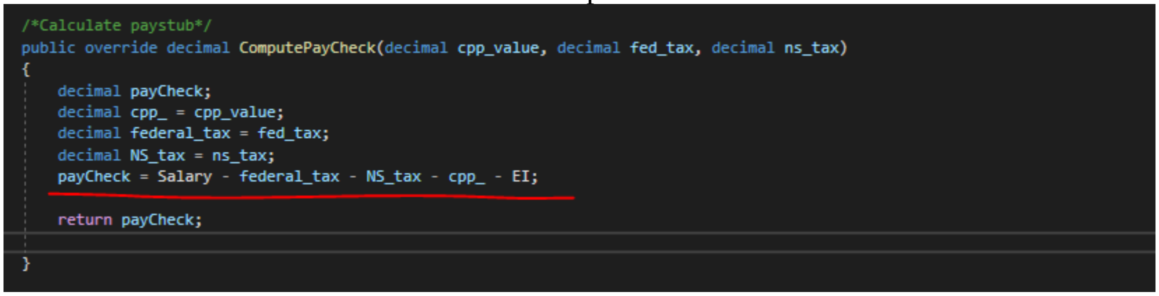
```
var employee = new Employee();
var average_Employee = new AverageEmployee("Yi", "Chen", "Dartmouth", 44000, 0.3m);
var programmer = new Programmer("Anna", "Sage", "Dartmouth", 83000, 0.9m);
var manager = new Manager("Amily", "Zhang", "Dartmouth", 150000, 0.5m);

// Creates and initializes a new ArrayList
```

- Make a loop to compute paycheck.

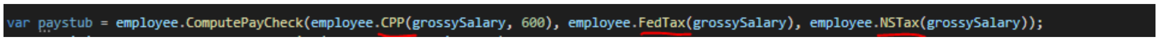


```
Console.Write("          " + "          ");
foreach (decimal riseRate in myARL)
{
    Console.Write("      |" + "        " + riseRate + "     ");
}
Console.WriteLine();
foreach (decimal grossySalary in myAL)
{
    Console.Write("        " + grossySalary + "        |");
    foreach (decimal riseRate in myARL)
    {
        var paystub = employee.ComputePayCheck(employee.CPP(grossySalary, 600), employee.FedTax(grossySalary), employee.NSTax(grossySalary));
        var payRising = employee.ComputePayRaise(grossySalary, riseRate);
        Console.Write("         " + payRising + "       |");
        /*Console.WriteLine("| paystub     |   "  + paystub + "     | pay rising       |   " + payRising  );*/
        /* Console.Write("      " + grossySalary + "      " +   paystub + "      "   +  riseRate    +"      " + payRising);*/
        /*Console.Write($"| grossy pay  | {grossySalary}      |    paystub     | { paystub}      | riseRate     | {riseRate} |  pay rised  |  {payRising} |" );*/
        /* Console.Write($"| Grossy salary    | {grossySalary}   | paystub is      | {paystub}       |");*/
        /* Console.WriteLine($"| Grossy salary    | {grossySalary}   | wage increased  | {payRising}          |");*/
    }
    Console.WriteLine("         ");
}
Console.WriteLine(" ______________________________________________________________________________      ");
Console.ReadLine();
```

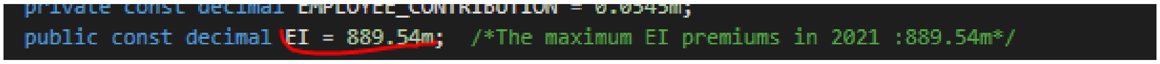
- Assume deductions such as taxes and pension.



```
/*Calculate paystub*/
public override decimal ComputePayCheck(decimal cpp_value, decimal fed_tax, decimal ns_tax)
{
    decimal payCheck;
    decimal cpp_ = cpp_value;
    decimal federal_tax = fed_tax;
    decimal NS_tax = ns_tax;
    payCheck = Salary - federal_tax - NS_tax - cpp_ - EI;

    return payCheck;

}
```

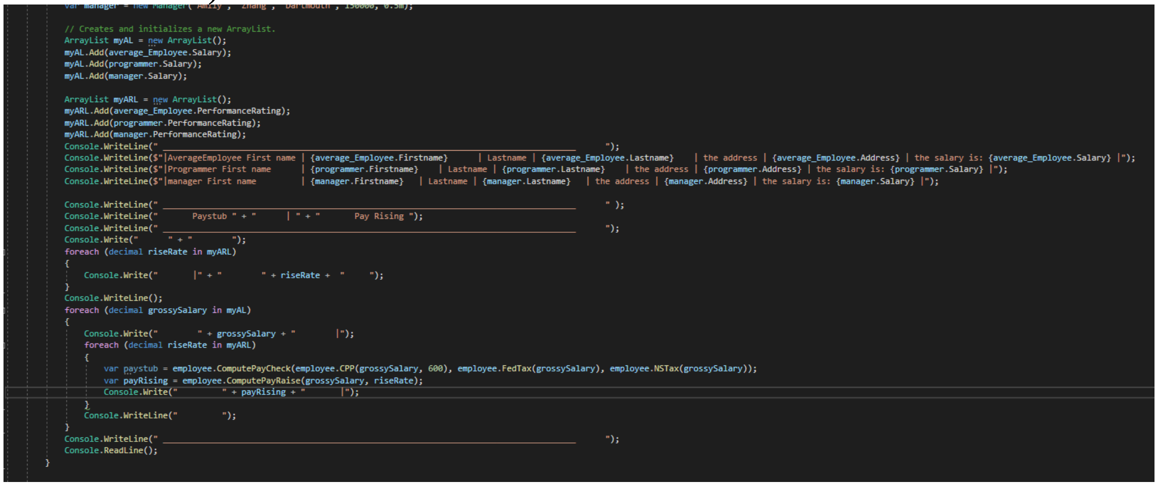


```
var paystub = employee.ComputePayCheck(employee.CPP(grossySalary, 600), employee.FedTax(grossySalary), employee.NSTax(grossySalary));
```



```
private const decimal EMPLOYEE_CONTRIBUTION = 0.0545m;
public const decimal EI = 889.54m;  /*The maximum EI premiums in 2021 :889.54m*/
```

- Write to console the paycheck summary in tabular format (information ordered in columns).



```
var manager = new Manager("Amily", "Zhang", "Dartmouth", 150000, 0.5m);

// Creates and initializes a new ArrayList.
ArrayList myAL = new ArrayList();
myAL.Add(average_Employee.Salary);
myAL.Add(programmer.Salary);
myAL.Add(manager.Salary);

ArrayList myARL = new ArrayList();
myARL.Add(average_Employee.PerformanceRating);
myARL.Add(programmer.PerformanceRating);
myARL.Add(manager.PerformanceRating);
Console.WriteLine(" ______________________________________________________________________      ");
Console.WriteLine($"|AverageEmployee First name | {average_Employee.Firstname}      | Lastname | {average_Employee.Lastname}    | the address | {average_Employee.Address} | the salary is: {average_Employee.Salary} |");
Console.WriteLine($"|Programmer First name      | {programmer.Firstname}    | Lastname | {programmer.Lastname}    | the address | {programmer.Address} | the salary is: {programmer.Salary} |");
Console.WriteLine($"|manager First name         | {manager.Firstname}   | Lastname | {manager.Lastname}   | the address | {manager.Address} | the salary is: {manager.Salary} |");

Console.WriteLine(" ______________________________________________________________________      " );
Console.WriteLine("      Paystub " + "      | " + "      Pay Rising ");
Console.WriteLine(" ______________________________________________________________________      ");
Console.Write("     " + "        ");
foreach (decimal riseRate in myARL)
{
    Console.Write("      |" + "        " + riseRate +  "     ");
}
Console.WriteLine();
foreach (decimal grossySalary in myAL)
{
    Console.Write("        " + grossySalary + "        |");
    foreach (decimal riseRate in myARL)
    {
        var paystub = employee.ComputePayCheck(employee.CPP(grossySalary, 600), employee.FedTax(grossySalary), employee.NSTax(grossySalary));
        var payRising = employee.ComputePayRaise(grossySalary, riseRate);
        Console.Write("         " + payRising + "       |");
    }
    Console.WriteLine("         ");
}
Console.WriteLine(" ______________________________________________________________________      ");
Console.ReadLine();
}
```

- Add a ReadLine() statement or some other method at the end of Main method to prevent the console app from terminating.

```
            Console.WriteLine("        ");
        }
        Console.WriteLine(" ____________________________________________________________________________________________ ");
        Console.ReadLine();
    }
```

3.3. Paycheck summary in tabular format

Table 2: Tabular Table

Write to console the paycheck summary in tabular format

```
C:\Users\Yi Chen\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe

_____________________________________________________________________________________________
|AverageEmployee First name | Yi      | Lastname | Chen    | the address | Dartmouth | the salary is: 44000 |
|Programmer First name      | Anna    | Lastname | Sage    | the address | Dartmouth | the salary is: 83000 |
|manager First name         | Amily   | Lastname | Zhang   | the address | Dartmouth | the salary is: 150000 |
_____________________________________________________________________________________________
      Paystub       |        Pay Rising
_____________________________________________________________________________________________
                    |        0.3            |        0.9            |        0.5
       44000        |         13200.0       |         39600.0       |         22000.0       |
       83000        |         24900.0       |         74700.0       |         41500.0       |
       150000       |         45000.0       |         135000.0      |         75000.0       |
_____________________________________________________________________________________________
```