C++ Workshop – 150018

Homework Assignment #6

Inheritance

1. At a college, there are two types of workers:
    - Full time employee (fulltime).
    - part-time workers (parttime).

We want to save the following data for each employee:

- First name of the employee
- ID of employee
- The employee's seniority at work (num of years)
- Percentage of salary that is paid for income tax

The payment policy at the college is once a month, according to the following data:

- For a full-time employee, the calculation is his annual salary divided by 12. (Therefore, for a full-time employee one should also keep his **annual salary**)
- For a part-time employee, the calculation is his monthly working hours times the payment promised to him for an hour of work. (Therefore, for a part-time employee both **working hours** and **hourly rate** should be saved.)

The college management decided to give each employee a bonus for the month of Tishrei, according to the following formula:

- If the employee has 5 years of experience or less, the bonus will be NIS 500.
- If the employee has more than 5 years of work experience, the bonus will be 25% of his monthly salary before the bonus.

Considering the above:

A. Define a basic class called **Employee**, which represents an employee at the college.

The class will include the following attributes:

- **name** - the employee's first name (string)
- **id** - ID of the employee (int)
- **seniority** - years of seniority of the employee (int)
- **pay** – monthly salary (float)

And the following methods:

- **constructor** for initializing the attributes (the pay attribute must be initialized to 0)
- Do not define a default-constructor in this class.
- **get** and **set** for setting and retrieving class attributes - if necessary
- **salaryAfterBonus** - to update the salary for the Tishrei bonus
- **Operator >>** for the insertion of employee data (the attributes must be received according to the order in which they are defined in the class) in the following format:

Enter employee details:

- **Operator <<** for printing the worker's details in the following format:

Employee:
Employee ID:
Years Seniority:

B. Define a class called **FullTime** that represents a full-time employee that inherits from the Employee class.
You must define the FullTime class so that the following methods can be run for an object in the class:
constructor - which receives the employee's details as parameters including annual salary (note, another field for this class) and initializes the fields with these values.
default constructor - with default values, empty string and zeros accordingly.
get and **set** for placing and retrieving class attributes - if necessary
salary - for calculating the employee's monthly salary
salaryAfterBonus - To update the salary for the Tishrei bonus
Operator >> For receiving employee data (the fields must be received according to the order in which they are defined in the class)
Enter employee details:
Operator << For printing employee data in the following format:

Employee:
Employee ID:
Years Seniority:
Salary per Month:

Pay attention! Code duplication should be avoided and no unused code should be written in this class. That is, if there is a method or part of the code in it - do not write another code snippet that performs the same operation !!

C. Similarly, define a class called **PartTime** that represents a part-time employee, and inherits from the Employee class.
You must configure the PartTime class so that the following methods can be run for an object in the class:
constructor - which receives as parameters the employee's details including working hours and hourly pay rate (note, additional attributes for this class) and initializes the attributes with these values.
default constructor - with default values, empty string and zeros respectively
get and set for setting and retrieving class attributes - if necessary
salary - for calculating the employee's monthly salary
salaryAfterBonus - To update the salary for the Tishrei bonus
Operator >> For receiving employee data (the fields must be received according to the order in which they are defined in the class)
Enter employee details:
Operator << For printing employee data in the following format:

Employee:
Employee ID:

Years Seniority:
Hours:

Salary per Month:

Pay attention! Code duplication should be avoided and no unused code should be written in this class. That is, if there is a method or part of the code in it - do not write another code snippet that performs the same operation !!

In any method in which an error may occur an exception ERROR should be thrown.
Please note that in the event of an error, despite the error, all employee data must be received.

Use the following main program to test your class:

```
#include "FullTime.h"
#include "PartTime.h"
#include <iostream>
using namespace std;
int main()
{
	FullTime arrF[3];
	for (int i = 0; i < 3; i++)
	{
		try
		{
			cin >> arrF[i];
		}
		catch (const char* str)
		{
			cout << str << endl;
			i--;
		}
	}


	PartTime arrP[3];
	for (int i = 0; i < 3; i++)
	{
		try
		{
			cin >> arrP[i];
		}
		catch (const char* str)
		{
			cout << str << endl;
			i--;
		}
	}


	for (int i = 0; i < 3; i++)
	{
		cout << arrF[i];
		cout << "After Bonus: " << arrF[i].salaryAfterBonus() << endl;
```

```
	}


	for (int i = 0; i < 3; i++)
	{
		cout << arrP[i];
		cout << "After Bonus: " << arrP[i].salaryAfterBonus() << endl;
	}
	return 0;
}
```

2. Define a new class **RoundList** which implements a circular linked list where the last link of the list points to the first link. **RoundList** should be able to perform all the methods that were defined for the linked list class as well as the following:
   - **addToEnd**(int val) . The method receives a whole number as a parameter, and adds the given value to the end of the list.
   - **search**(int n) . The method receives a whole non negative number, n, the method returns the value found in the nth location in the list. Note, n may be larger than the number of elements in the list. In this case, the search wraps around to the beginning of the list and continues its search until it finds the element with index n. The index of the first element of the list is 0. In the case of an empty list, the function returns -1.

   The RoundList class should inherit from the List class defined on page 17 in the course booklet or the one seen in class. Implement all the necessary methods from List as well as the two additional methods for a Circular list.

   Additional assumptions:
   - Do not add any new private fields to the RoundList class. The only field in the class should be the head pointer as defined in the List class.
   - Decide which methods in the List class should be overloaded by the RoundList class and which methods should not.

   Use the following main program to test your class:

```
#include "RoundList.h"
#include <iostream>
using namespace std;

enum CHOICES{
	EXIT, ADD, ADD_TO_END, REMOVE_FIRST, SEARCH, CLEAR, EMPTY
};
int main(){

	RoundList ls1;
	int choice;
	cout << "Enter your choice: ";
	cin >> choice;
	while(choice != EXIT)
```

```
	{
		int num;
		switch(choice){
			case ADD :	cout << "Enter 5 numbers: ";
					for(int i=0; i < 5; i++)
					{
						cin >> num;
						ls1.add(num);
					}
					break;

			case ADD_TO_END :cout << "Enter 5 numbers: ";
						for(int i=0; i < 5; i++)
						{
							cin >> num;
							ls1.addToEnd(num);
						}
						break;

			case REMOVE_FIRST : ls1.removeFirst();
						break;

			case SEARCH: cout << "Enter a number: ";
					cin >> num;
					cout << ls1.search(num)<<endl
							break;

			case CLEAR: ls1.clear();
					break;

			case EMPTY: if(ls1.isEmpty())
						cout << "Empty"<<endl;
					else
						cout << "Not empty" << endl;
					break;

			default: cout<< "ERROR!"<<endl;
		}
		cout << "Enter your choice: ";
		cin >> choice;
	}
	return 0;
}
```

An example of running the program: