**C++ Workshop – 150018**

**Homework Assignment #6**

**Inheritance**

**Question 1**

At a college, there are two types of employees:

* Full time employees (fulltime).
* Part time employees (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 (number of years)
* The employee’s monthly earnings

The college pays its employees once a month according to the following policy:

* For a full-time employee, the calculation is his annual salary divided by 12. (Therefore, for a full-time employee the class will need to store the employee’s **annual salary**)
* For a part-time employee, the calculation is the number of hours that he worked for the month times the amount that he is paid per hour. (Therefore, for a part-time employee both **working hours** and **hourly rate** need to be stored in the class.)

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

**Part A:**

Define the base class **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 constructors/methods:

* **constructor** for initializing the attributes of **name, id,** and **seniority** according to the values that were passed as parameters, **pay** must be initialized to 0,
* **empty constructor** – will give default values to all of the attributes (0s and empty strings)
* **copy constructor**
* **get** and **set** for setting and retrieving class attributes - if necessary
* **salaryAfterBonus** - to update the salary for the Tishrei bonus

(This function will only be defined in the base class. You will not override it in the derived classes (aka you will only define it in the class Employee.)

* **operator >>** for the input 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 out the worker's details according to the following format:

Employee:   
Employee ID:   
Years Seniority:

**Part** **B:**

Define a class called **FullTime** that represents a full-time employee that inherits from the Employee class.

The class will include the following attributes (in addition to the attributes that are already defined in Employee)

* salaryPerYear – the employee’s annual salary (float)

And the following constructors/methods:

* **constructor** - which receives the employee's details as parameters including annual salary (note, an additional field for this class) and initializes the fields with these values. The constructor does not receive the value for the parameter **pay,** but rather will call the function salary (details below) to initialize the variable **pay.**
* **empty constructor** – will give default values to all of the attributes (0s and empty strings).
* **copy constructor**
* **get** and **set** for placing and retrieving class attributes - if necessary
* **salary** - for calculating the employee's monthly salary
* ***Food for Thought : Why don’t we need to define the method* salaryAfterBonus**? ***What will happen at runtime?***
* **operator >>** for the input 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:

Reminder: the program does not receive the value for the attribute **pay** as input but rather calls the method **salary** to calculate **pay**’s value.

**operator <<** for printing out the worker's details according to the following format

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

**Pay attention! Avoid writing unwanted, unused, and duplicate code. In other words, if there is a method that already exists in the parent class then there is no need to rewrite it in the child class!!**

**Part C:**

Similarly, define a class called **PartTime** that represents a part-time employee, and inherits from the Employee class.

The class will include the following attributes(in addition to the attributes that are already defined in Employee)

* hoursOfWork – the number of hours that the employee worked for the mont (float)
* payPerHour – the employee’s hourly wage

And the following constructors/methods:

* **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. The constructor does not receive the value for the parameter **pay,** but rather will call the function salary (details below) to initialize the variable **pay.**
* **empty constructor** – will give default values to all of the attributes (0s and empty strings).
* **copy constructor**
* **get and set** for setting and retrieving class attributes - if necessary
* **salary** - for calculating the employee's monthly salary
* ***Food for Thought : Why don’t we need to define the method* salaryAfterBonus**? ***What will happen at runtime?***
* **operator >>** for the input 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:

Reminder: the program does not receive the value for the attribute **pay** as input but rather calls the method **salary** to calculate **pay**’s value.

**operator <<** for printing out the worker's details according to the following format:

Employee:  
Employee ID:   
Years Seniority:   
Hours:

Salary per Month:

**Pay attention! Avoid writing unwanted, unused, and duplicate code. In other words, if there is a method that already exists in the parent class then there is no need to rewrite it in the child class!!**

In any method in which an error may occur an exception ERROR should be thrown.

Please note that in the event of an error, all employee data is first read before the error is thrown.

Use the following main program to test your class:

#include "FullTime.h"

#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;

}

An example of the program with 3 employees (the two first are full time and the third is part time).

Enter employee details:

moshe 1234 3 36000

Enter employee details:

miriam 4321 8 48000

Enter employee details:

aharon 5678 4 160 35

Employee: moshe

Employee ID: 1234

Years Seniority: 3

Salary per Month: 3000

After Bonus: 3500

Employee: miriam

Employee ID: 4321

Years Seniority: 8

Salary per Month: 4000

After Bonus: 5000

Employee: aharon

Employee ID: 5678

Years Seniority: 4

Hours: 160

Salary per Month: 5600

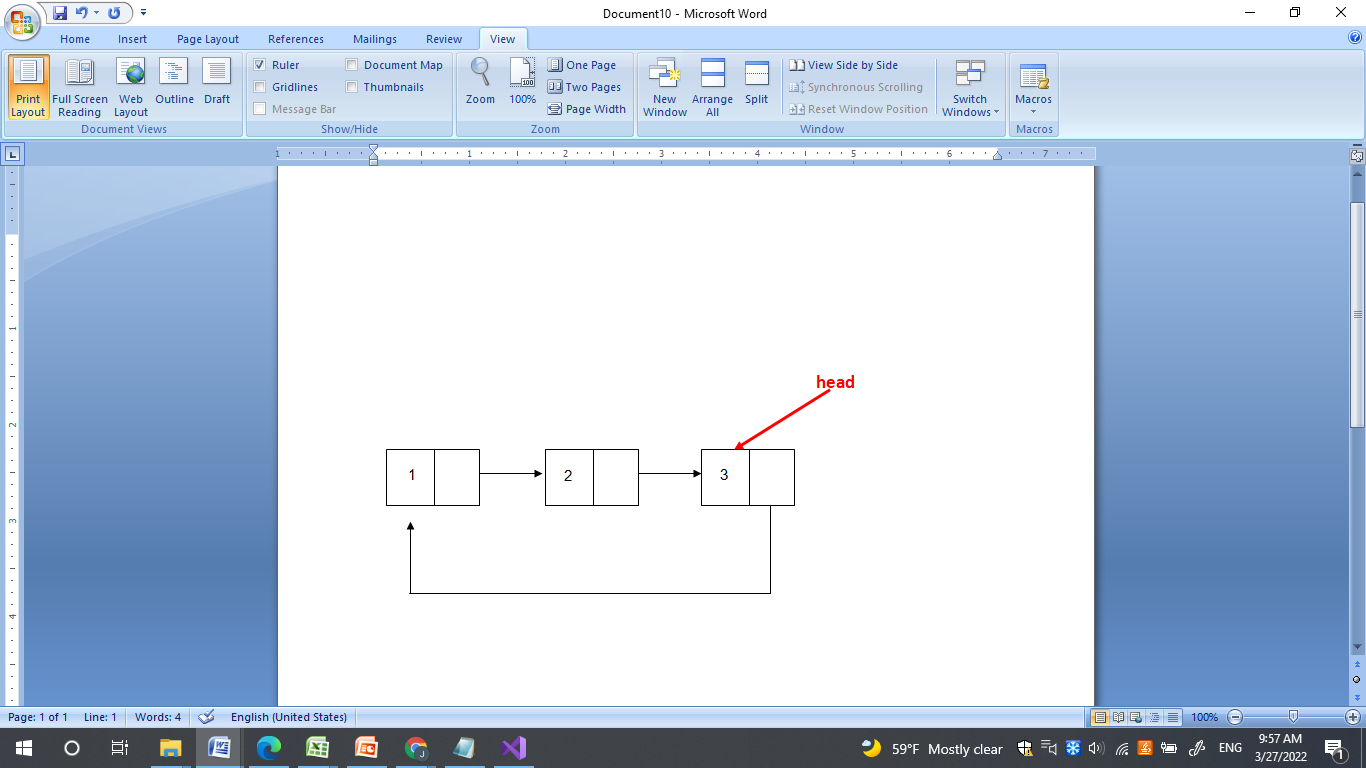
**Question 2**

Define a new class **RoundList** which implements a circular linked list where the last link of the list points to the first link.

For example, consider the Roundlist that has 3 elements

* the first element is 1
* the second element is 2
* the third element is 3

The list would appear as follows:



In other words, the first element points to the second element, the second element points to the third element, and the third (last) element points to the first element. Head points to the last element in the list

There is only one head

**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.
* **operator <<** to print out the elements that are stored in the list. It must start the printing at the first element and after each element it should print a blank space. If the list is empty, then it shouldn’t print anything.

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 three additional methods for the RoundList.

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 need to have an override and which methods should not.
  + Reminder: the constructor of RoundList will call the constructor of List and the destructor of RoundList will call the destructor of List. Keep this in mind when writing the constructors and destructor of RoundList

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, PRINT

};

int main() {

RoundList roundList;

int choice;

cout << "Enter your choice: \n";

cin >> choice;

while (choice != EXIT) {

int num;

switch (choice) {

case ADD:

cout << "Enter a number: \n";

cin >> num;

roundList.add(num);

break;

case ADD\_TO\_END:

cout << "Enter a number: \n";

cin >> num;

roundList.addToEnd(num);

break;

case REMOVE\_FIRST:

roundList.removeFirst();

break;

case SEARCH:

cout << "Enter a number: ";

cin >> num;

cout << roundList.search(num) << endl;

break;

case CLEAR:

roundList.clear();

break;

case EMPTY:

if (roundList.isEmpty())

cout << "Empty" << endl;

else

cout << "Not empty" << endl;

break;

case PRINT:

cout << roundList << endl;

break;

default: cout << "ERROR!" << endl;

}

cout << "Enter your choice: \n";

cin >> choice;

}

return 0;

}

An example of a running program

Enter your choice:

1

Enter a number:

10

Enter your choice:

1

Enter a number:

11

Enter your choice:

7

11 10

Enter your choice:

2

Enter a number:

20

Enter your choice:

2

Enter a number:

21

Enter your choice:

7

20 21 11 10

Enter your choice:

6

Not empty

Enter your choice:

3

Enter your choice:

7

21 11 10

Enter your choice:

4

Enter a number:

1

11

Enter your choice:

5

Enter your choice:

6

Empty

Enter your choice: