VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING FUNDAMENTALS - CO1027

ASSIGNMENT 2

SHERLOCK
A STUDY IN PINK - Part 3

HO CHI MINH CITY, MARCH 2022

ASSIGNMENT SPECIFICATIONS

Version 1.0

1 Outcomes

After finishing this assignment, the student is revised and can proficiently use:

- Pointer
- Object-oriented programming (OOP)
- Singly linked list

2 Introduction

The assignment is adapted from Episode 1, Season 1 of the BBC series Sherlock. This movie is also derived from the Sherlock Holmes novel by author Sir Arthur Conan Doyle.

At the end of part 2, a taxi driver appeared in front of apartment number 221B Baker Street and invited Sherlock to be a guest on the next ride. Sherlock knows that this driver is a criminal, but, Sherlock still cannot understand, why the victims after a taxi ride committed suicide. Sherlock has chosen to join this dangerous trip.

On the other hand, after a while Sherlock left the room, Watson checked the position signal again on his laptop. He noticed that the location of the phone was slowly moving away from the apartment. Watson also took another bus that followed the location of the phone.

In this assignment, you are asked to implement the following Classes:

1. **class Person**: class represents a character, e.g. the criminal, Sherlock, Watson.
2. **class Path**: class to store the points that a character passes through, so in the Person class, there will be a variable of type Path to store these points.
3. **class Node**: class represents a Node in class Path, class Path will be implemented based on *Singly linked list*. Each element of the list is a Node with 2 attributes as the data to be stored, and a link (pointer) to the next Node.
4. **class Point**: class represents a point (position) during the movement of characters. This point is the data of a Node.

The details of the classes to be implemented will be detailed in the tasks below. We will be

implementing the classes in increasing complexity, so the tasks below will require implementing the above classes in the reverse order: Point, Node, Path, Person.

3 Tasks 1

Students are asked to implement a program in C++ to simulate the solving process of the first case of Sherlock and Watson: A study in Pink, through the tasks described below. Each task is required to implement the methods (methods) of the respective class. The parameters for these methods will be given in the description of the task request.

3.1 Task 1: Implement class Point (1.5 points)

The definition of the class Point is given as:

```
class Point {
private:
    int x;
    int y;
public:
    Point(int x=0, int y=0);
    string toString() const;

    int distanceTo(const Point & otherPoint) const;
};
```

In which: the x, y attributes are the positions along the Ox and Oy axes in the coordinate plane, respectively. We do not need to care about the specific direction of the 2 axes in this assignment.

1. Implement the object initialization method (Constructor):

   ```
       Point(int x=0, int y=0);
   ```

   This method assigns the values of the x, y parameters to the corresponding x, y properties of the class. Parameters x, y are passed as default parameters.
2. Implement the **toString** method:

   ```
       string toString() const;
   ```

   This method returns a string representation of the current object's value. See below example of string representation:

Example 3.1

Example of string representation for class Point. Given the following code:

```
Point p1(0, 5);
cout << p1.toString() << endl;
```

Print results to the screen:

```
<Point[0,5]>
```

Example 3.2

Example of string representation for class Point. Given the following code:

```
Point p2(9, 3);
cout << p2.toString() << endl;
```

Print results to the screen:

```
<Point[9,3]>
```

3. Implement the **distanceTo** method:

```
int distanceTo(const Point & otherPoint) const;
```

The method returns the Euclidean distance from the current object to the **otherPoint** object. The Euclidean distance between two points p1(x1,y1) and p2(x2,y2) is calculated by the formula:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

If there is decimal part in d then we round up the value of d. The distanceTo method returns this d value.

Example 3.3

Given the following code fragment:

```
Point p3(0, 1);
Point p4(2, 2);
cout << p3.distanceTo(p4) << endl;
```

Print results to the screen:

```
3
```

Explanation:

$$d = \sqrt{2^2 + 1^2} = \sqrt{5} \approx 2.24 \xrightarrow{\text{Round up}} 3$$

3.2 Task 2: Implement class Node (1.5 points)

The definition of class Node is given as:

```
1  class Node {
2  private:
3      Point point;
4      Node * next;
5
6      friend class Path;
7
8  public:
9      Node(const Point & point=Point(0,0), Node * next=NULL);
10     string toString() const;
11 };
```

In which:

- **point** attribute: contains the data that this Node object stores, which data type is Point.
- **next** attribute: pointer contains the address of the next Node in Path.
- Line 6: declare the class Path as a friend to the class Node so that we can easily access 2 properties of Node while in the Path implementation.

Students are asked to implement the following methods of the class Node:

1. **Node** Constructor:

```
1      Node(const Point & point=Point(0,0), Node * next=NULL);
```

- Input parameters:
  - **point**: contains the value of the point to be assigned to the *point* property of object.
  - **next**: contains the value to be assigned to the *next* property of object.
- Requirements: Initialize the corresponding properties as described in *Input parameters*.

2. **toString** method:

```
    string toString() const;
```

- Input parameters: none.
- Output results: string representing the data value of a Node object. Refer to the examples below for string representation:

Example 3.4

Given the following code fragment:

```
    Node node1(Point(1, 2));
    Node node2(Point(2,3), &node1);
    cout << "node 1:" << node1.toString() << endl;
    cout << "node 2:" << node2.toString() << endl;
```

Print results to the screen:

```
node␣1:<Node[<Point[1,2]>]>
node␣2:<Node[<Point[2,3]>]>
```

Attention: Pay attention to the string representing Point in the above example, is it possible to reuse the toString function of the class Point while implementing this function?

3.3 Task 3: Implement class Path (3 points)

The definition of class Path is given as:

```

class Path {
private:
    Node * head;
    Node * tail;
```

```
    int count;
    int length;

public:
    Path();
    ~Path();

    void addPoint(int x, int y);
    string toString() const;
    Point getLastPoint() const;
};
```

In which:

- **head** attribute: a pointer containing the address of the first Node in the Path. If the Path object does not have any Nodes, then **head** has the value of **NULL**.
- **tail** attribute: a pointer containing the address of the last Node in the Path. If the Path object does not have any Nodes, then **head** has the value of **NULL**.
- **count** attribute: stores the number of Nodes currently in the Path.
- **length** attribute: stores the current length of the Path. If Path has no Nodes then **length** has a value of -1. If Path has only 1 Node then **length** has value 0. If Path has k Nodes with $k >= 2$, call the data that k Nodes is storing as k points: $p_1, p_2, ..., p_k$. Then **length** is calculated as follows:

$$\text{length} = \lceil d(p_1, p_2) \rceil + \lceil d(p_2, p_3) \rceil + ... + \lceil d(p_{k-1}, p_k) \rceil$$

With:

  - $\lceil x \rceil$: is the rounding operation of x to the nearest integer (including itself).
  - $d(p_m, p_n)$: s the Euclidean distance between 2 points p_m and p_n as stated in Task 1.

Students are asked to implement the following methods of the class Path:

1. Path Constructor:

```
    Path();
```

   - Input parameters: none.
   - Requirements: initialize values for properties as described above.

2. Path Destructor:

```
~Path();
```

- Input parameters: none.
- Requirements: free all memory that allocated dynamically (if any) by this object.

3. **addPoint** method:

```
void addPoint(int x, int y);
```

- Input parameters:
  - **x**: position on the Ox axis of the point to be added.
  - **y**: position on the Oy axis of the point to be added.
- Requirements:
  Create a new point named **x, y** and add it to the end of the list maintained by **head**, **tail** in the class Path. Note: Path properties must be updated to maintain what was described above.
- Output results: none.

4. **toString** method:

```
string toString() const;
```

- Input parameters: none.
- Output results: string representing the data value of an object of class Path. Refer to the examples below for string representation.

Example 3.5

Given the following code fragment:

```
Path p1;
cout << p1.toString() << endl;
p1.addPoint(0, 1);
cout << p1.toString() << endl;
p1.addPoint(1, 3);
cout << p1.toString() << endl;
```

Print results to the screen:

```
<Path[count:0,length:-1,[]]>
<Path[count:1,length:0,[<Node[<Point[0,1]>]>]]>
<Path[count:2,length:3,[<Node[<Point[0,1]>]>,<Node[<Point[1,3]>]>]]>
```

Attention: Can the string representation for class Node be reused in the **toString** function of this class Path?

5. **getLastPoint** method:

```
Point getLastPoint() const;
```

- Input parameters: không.
- Output results: returns the Point object corresponding to the data located at the last Node of the Path. (Testcase will make sure Path always has at least 1 Node before calling this method.)

Example 3.6

Given the following code fragment:

```
Path p2;
p2.addPoint(0, 1);
Point pt1 = p2.getLastPoint();
cout << pt1.toString() << endl;

p2.addPoint(5, 6);
Point pt2 = p2.getLastPoint();
cout << pt2.toString() << endl;
```

Print results to the screen:

```
<Point[0,1]>
<Point[5,6]>
```

3.4 Task 4: Implement class Character (3 points)

The class Character represents a character in this assignment. A character is indicated by his/her name and the points where he/she move. We will store these travel points in an object of class Path. The class Character definition is given as follows:

```
class Character {
private:
    string name;
    Path * path;

public:
    Character(const string & name="");
    ~Character();

    string getName() const;
    void setName(const string & name);

    void moveToPoint(int x, int y);
    string toString() const;
};
```

In which:

- **name** attribute: string representing the character's name.

- **path** attribute: pointer of type Path, containing the address of an object of class Path. This attribute is used to store the points that this character passes through.

Students are asked to implement the following methods of the class Character:

1. Character Constructor:

```
Character(const string & name="");
```

   - Input parameters:
     - **name**: string representing the name to be assigned to the **name** property of the object.
   - Requirements:
     - Initializes the **name** property value with the value of the **name** variable.
     - Initializes an object of class Path (by dynamic allocation) and assigns its value to the **path** property.

2. Character Destructor:

```
~Character();
```

   - Input parameters: none.
   - Requirements: free all memory that allocated dynamically (if any) by this Character object.

3. **getName** method:

```
string getName() const;
```

   - Input parameters: none.
   - Output results: Returns the string corresponding to the name of that stored in Character object.

4. **setName** method:

```
void setName(const string & name);
```

   - Input parameters:
     - **name**: string containing the name to be assigned to the **name** property of the Character object.
   - Requirements: assigns a name to the **name** property of the Character object with the value passed.

- Output results: none.

Example 3.7

Given the following code fragment:

```
Character chWatson("Watson");
cout << chWatson.getName() << endl;
chWatson.setName("John Watson");
cout << chWatson.getName() << endl;
```

Print results to the screen:

```
Watson
John Watson
```

5. **moveToPoint** method

```
void moveToPoint(int x, int y);
```

- Input parameters:
  - **x**: position along the Ox axis of the point that this Character object will move to.
  - **y**: position along the Oy axis of the point that this Character object will move to.
- Requirements: add a new point at the end of the **path** attribute with the x, y position corresponding to the value passed. This process represents that this Character object has moved to a new point, and this point is stored at the end of the **path** property.
- Output results: none.

6. **toString** method:

```
string toString() const;
```

- Input parameters: none.
- Output results: string representing the value of the Character object. Refer to the example below for string representation.

Example 3.8

Given the following code fragment:

```
Character chWatson("Watson");
cout << chWatson.toString() << endl;
chWatson.moveToPoint(2, 7);
cout << chWatson.toString() << endl;
```

Print results to the screen:

```
<Character[name:Watson,path:<Path[count:0,length:-1,[]]>]>
<Character[name:Watson,path:<Path[count:1,length:0,[<Node[<Point[2,
↪ 7]>]>]]>]>
```

3.5 Task 5: Rescue Sherlock (1 points)

After putting Sherlock in a taxi, the criminal drove him through many streets, eventually stopping at a school. He reveals how to carry out previous cases to Sherlock. The criminal will take out 2 potions, one is poison, the other is normal medicine. The victim has to choose one potion and the criminal chooses the other. Sherlock thinks this is simply random luck. But the criminal didn't think so, there were 4 potion selections happening, and he won all 4 times. For him, this is the talent of predicting and manipulating the victim to choose the potion. This time, the criminal invites Sherlock, the detective who always confident with his excellent logical reasoning ability, to join the game.

On the other hand, Watson is also following the position of the locator, but this path is not the same as that of Sherlock. Watson could be taken a further route. If this route was too far he wouldn't be able to get to Sherlock in time. If this path brings him to a location close enough to Sherlock, Watson can use his knack for shooting and shoot at the criminal. Sherlock is also startled by the sound of gunfire and does not take the selected pill.

Students are asked to write the following function to describe the above process:

- Function name: **rescueSherlock**
- Function declaration:

```
bool rescueSherlock(
const Character & chMurderer,
const Character & chWatson,
```

```
4         int maxLength,
5         int maxDistance,
6         int & outDistance
7         );
8
```

- Input parameters:
  - **chMurderer**: the Character object represents the criminal.
  - **chWatson**: the Character object represents Watson.
  - **maxLength**: the maximum length that Watson's path length is allowed to exceed the criminal's path length.
  - **maxDistance**: maximum distance between Watson's location and the criminal.
  - **outDistance**: variable passed by reference to return the distance between Watson and the criminal.
- Requirements: let l_1, l_2 be the path lengths of Watson and of the criminal, respectively.
  - If $l_1 - l_2$ <= maxLength: assign **outDistance** equal to the distance between Watson and the criminal. If this distance does not exceed **maxDistance** then Watson hits the criminal and successfully rescues Sherlock. In contrast, Watson failed to rescue Sherlock.
  - If $l_1 - l_2$ > maxLength: assign **outDistance** to -1 and Watson fails to rescue Sherlock in this case.
- Output results: returns **true** if Watson successfully rescues Sherlock. Otherwise, returns **false.**

Attention: You may need to write some helper methods in the given classes to accomplish this task.

3.6 Task 6:

Note:

- This is an exclusive and mandatory requirement for the students of the Course **Programming Engineering Project**, the students of the Course **Programming Fundamentals** please ignore this requirement.
- The formula score for students of Course **Programming Engineering Project** is:

Total scores = (Scores for Task 1-5) * 0.7 + (Scores for Task 6) * 0.3

Students are required to implement the following two methods of class **Path**:

1. **removeLast** method:

```
bool removeLast();
```

- Input parameters: none.
- Requirements: delete the last Node of the Path object, students need to pay attention to update the properties of the object with the value that matches the description of class Path.
- Output results: returns **true** if the last Node can be removed. Otherwise, returns **false**.

Example 3.9

Given the following code fragment:

```
Path p1;
p1.addPoint(0, 1);
p1.addPoint(1, 2);
cout << p1.toString() << endl;
p1.addPoint(3, 5);
p1.removeLast();
cout << p1.toString() << endl;
```

Print results to the screen:

```
<Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>]]>
<Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>]>]]>
```

2. Overload the addition operator (+) to be able to perform addition on 2 Path objects.

```
Path operator+ (const Path & other);
```

- Input parameters: (self-research).
- Requirements: instantiates a new Path object by concatenating two Path objects.
- Output results:returns a new Path object. Note that this object needs a separate memory area for each of its properties.

Example 3.10

Given the following code fragment:

```
Path p1;
p1.addPoint(0, 1);
p1.addPoint(1, 2);

Path p2;
p2.addPoint(3, 4);

Path p3 = p1 + p2;

cout << "p1: " << p1.toString() << endl;
cout << "p2: " << p2.toString() << endl;
cout << "p3: " << p3.toString() << endl;

Path p4;
p4.addPoint(0, 1);
p4.addPoint(1, 2);
p4.addPoint(3, 4);
cout << "p4: " << p4.toString() << endl;
```

Print results to the screen:

```
p1: <Path[count:2,length:2,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>
↪ ]>]]>
p2: <Path[count:1,length:0,[<Node[<Point[3,4]>]>]]>
p3: <Path[count:3,length:5,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>
↪ ]>,<Node[<Point[3,4]>]>]]>
p4: <Path[count:3,length:5,[<Node[<Point[0,1]>]>,<Node[<Point[1,2]>
↪ ]>,<Node[<Point[3,4]>]>]]>
```

3.7 Ending

Watson arrives just in time and fires a shot from a distance to stop Sherlock from taking the pill. Sherlock looks through the bullet marks, he sees the accuracy and experience of a man from the battlefield. Sherlock looked at his partner Watson approaching. He said police no longer need to investigate the shooter. Sherlock knows, he just got help!!

Thank you for accompanying Sherlock and Watson in this semester's assignment. I wish

you all the best and good luck in your studies!!!

4 Submission

Students submit a file: **studyInPink3a.h** in the site "Ky thuat lap trinh (CO1027)_HK212_ALL". Students of the Programming Fundamentals **Project** course submit 1 file: **studyInPink3b.h**.

Deadlines for submission are announced at the submission site above. By the deadline for submission, the link will be locked automatically, so students will not be able to submit them late. To avoid possible risks at the time of submission, students MUST submit their papers at least **one hour** before the deadline.

5 Handling fraud

Assignment must be done BY YOURSELF. Students will be considered fraudulent if:

- There is an unusual similarity between the source code of the submissions. In this case, ALL submissions are considered fraudulent. Therefore, students must protect the source code of their assignments.
- Students do not understand the source code written by themselves, except for the parts of the code provided in the initialization program. Students can consult from any source, but make sure they understand the meaning of all the lines they write. In the case of not understanding the source code of the place they refer, students are especially warned NOT to use this source code; instead use what has been learned to write programs.
- Mistakenly submit another student's assignment on your personal account.

In the case of cheating, students will get a 0 for the entire subject (not just the assignment).

DO NOT ACCEPT ANY INTERPRETATION AND NO EXCEPTION!

After each major assignment has been submitted, a number of students will be called for random interviews to prove that the assignment has been done by themselves.

6 Change from previous version

References

[1] A Study in Pink, Wikipedia, https://en.wikipedia.org/wiki/A_Study_in_Pink

[2] Sherlock, Season 1 - Episode 1: A Study in Pink, Netflix, https://www.netflix.com/watch/70174779?trackId=13752289.

END