Class Tour

A diagram of a circle with different colored squares

Description automatically generated

**Implementing Class Tour** (85 points)

You are given the following:

* Class **Node** in **tour.cpp**. This class represents a node in a doubly-linked list.
* An incomplete implementation of class **Tour**. This class represents a circular tour as shown in the illustrations above.
* Class **Point2D** in **point.h**. Do ***not*** change this file.

Implement the following functions.

// [6 points] Inserts a new node with the given

// value after the given node.

// Throws a string exception if node is a nullptr.

void insert\_after(Node<T>\* node, const T& val);

// [6 points] Inserts a new node with the given

// value before the given node.

// Throws a string exception if node is a nullptr.

void insert\_before(Node<T>\* node, const T& val);

// [6 points] Adds a new node with the given value.

// The newly added node becomes the last node in the tour.

// (the node before start)

void add\_to\_end(const T&);

// [6 points] adds a new node with the given value.

// The newly added node becomes the start of the tour.

// The old start node becomes the 2nd node in the tour.

void add\_to\_start(const T&);

// [25 points] Removes all the nodes in the tour between ptr1 and ptr2 (exclusive).

// Check the illustration provided below for a more complete description.

// Throws a string exception if ptr1 or ptr2 is nullptr.

// Throws a string exception if ptr1 and ptr2 point to nodes that are not

// in the same tour.

void cut(Node<T>\* ptr1, Node<T>\* ptr2);

// [6 points] Removes all the nodes in the tour.

// (This function must use function cut());

void clear();

// [15 points] Check if the tour is symmetric. I.e. if visiting

// the nodes from start node to the last node (the right before the start node)

// produces the same sequence of "values" for visiting the nodes starting from

// the last one and back to the start node.

// Consider an empty tour and a tour with one node as symmetric.

// Example.

// the following are symmetric tours (of Point2D objects):

// (1, 2) --> (2, 5) --> (1, 2)

// (1, 2) --> (2, 5) --> (5, 8) --> (2, 5) --> (1, 2)

// (1, 2) --> (2, 5) --> (5, 8) --> (5, 8) --> (2, 5) --> (1, 2)

// the following are not symmetric tours:

// (1, 2) --> (2, 5)

// (1, 2) --> (2, 5) --> (5, 8) --> (2, 5)

// (1, 2) --> (2, 5) --> (5, 8) --> (4, 6) --> (2, 5) --> (1, 2)

bool is\_symmetric\_tour() const;

// [15 points] Computes the length of a tour of Point2D objects.

// The length of the tour is defined as:

// length = dist(start\_node, 2nd\_node) + dist(2nd\_node, 3rd\_node) + ... +

// dist(last\_node, start\_node)

// Example.

// (0, 0) --> (1, 1) --> (2, 2) --> (3, 3)

// length = dist((0,0), (1,1)) + dist((1,1), (2,2)) +

// dist((2,2), (3,3)) + dist((3,3), (0,0))

// = sqrt(2) + sqrt(2) + sqrt(2) + sqrt(18)

// = 8.48528137424

// Return 0 if the tour is empty or contains 1 point.

friend double tour\_length(const Tour<Point2D>& tour);

We *recommend* that you implement the functions in the same order provided above.

You are allowed to add new **private** functions, but you are ***not*** allowed to add new **public** functions or change the signature of the given public functions.

**Cutting Part of the Tour**

The following illustrations explain how function **cut**(Node<T>\* ptr1, Node<T>\* ptr2) should behave.

**1.** General case

Remove all the nodes between ptr1 and ptr2 (excluding ptr1 and ptr2) as the following illustration shows.

A diagram of a process

Description automatically generated

**2.** If **start** is in the cut range.

Make start be ptr1 as the following illustration shows.

A diagram of a process

Description automatically generated

**3.** If **ptr1 == ptr2**

Remove all the nodes in the tour except the one pointed at by ptr1 and ptr2. Also, make start point at ptr1. The following is an example.

A diagram of a circular diagram

Description automatically generated with medium confidence

**4.** Special cases

Throw an exception of type string with a suitable error message in each of the following cases:

* If ptr1 or ptr2 is nullptr.
* If ptr1 or ptr2 point at nodes that are not in the same tour.

**Running Time**

The following functions must run in (1)*O*(1):

* insert\_before(...)
* insert\_after(...)
* add\_to\_end(...)
* add\_to\_start(...)

The following functions must run in *O*(*n*) (where *n* is the number of nodes in the tour):

* cut(...)
* clear()
* tour\_length(...)
* is\_symmetric()

**Testing Class Tour** (13 points)

Test your implementation by writing a main() program in main.cpp. Your program must:

* Call every **public** function in class Tour at least once in a meaningful way.
* Call every **friend** function of class Tour at least once in a meaningful way.

To test your code using your main method, click on the **Run** button.

**Note.** To test the **is\_symmetric\_tour**() function, use integers (not **Point2D** objects, because no comparison operators are defined for this type).

assignment. It is your responsibility to test your code and make sure that it is correct and compliant with the assignment requirements.

You are allowed to click the **Submit** button and use the automatic grader as many times as you like *before the deadline.*

**Note.** If your insertion functions are buggy, you will most probably fail all the test cases of all the other functions.