C++ Test

Nikolaj Roager Christensen

nikolaj@ulvedalen.dk

Result

Status: PASS

Completed: Jun 12

Time: 43min

Total:

90%

Top 5% of all candidates

Breakdown

C++ 90%

Question 1

C++ . LINKED LIST . PREMIUM

Score: 100%

Time: 23min 1sec

Difficulty: Easy

Description:

You are holding one link of a chain in your hand. Implement method *longerSide* to find which side of the chain, relative to the link you are holding, has more links.

If the left side has more links return *Side::LEFT*, if the right side has more links return *Side::RIGHT*, and if both sides have an equal number of links or if the chain is a closed loop, return *Side::NONE*. For example, for the code below, the output should be *Side::RIGHT* (or 2):

```
ChainLink* left = new ChainLink();
ChainLink* middle = new ChainLink();
ChainLink* right = new ChainLink();
left->append(middle);
middle->append(right);
std::cout << left->longerSide();
```

Answer

```
#include <stdexcept>
#include <iostream>

enum Side { NONE, LEFT, RIGHT };

class ChainLink
{
public:
```

```
    void append(ChainLink* rightPart)
    {
        if (this->right != NULL)
            throw std::logic_error("Link is already connected.");

        this->right = rightPart;
        rightPart->left = this;
    }

    Side longerSide()
    {
        //We have to traverse the chain left and right, it is easy, since it is a double linked list
        //We should however make sure we are not in a loop

        int size_left=0;
        int size_right=0;


        //Just loop around, and check if we get back
        for(ChainLink* Link=this; Link!=NULL; Link=Link->right)
        {
            if (size_right!=0 && Link==this)//We could use a set to store all visited, but this works too
                return Side::NONE;
            ++size_right;
        }

        //Just loop around, and check if we get back
        for(ChainLink* Link=this; Link!=NULL; Link=Link->left)
        {
            //No need to check this, we would have caught that when going right
            //if (size_left!=0 && Link=this)
            //    return Side::NONE;
            ++size_left;
        }

        return (size_right==size_left ? Side::NONE : (size_right>size_left ? Side::RIGHT:Side::LEFT));


    }

private:
    ChainLink* left;
    ChainLink* right;
};

#ifndef RunTests
int main()
{
    ChainLink* left = new ChainLink();
    ChainLink* middle = new ChainLink();
    ChainLink* right = new ChainLink();
    left->append(middle);
    middle->append(right);

    std::cout << left->longerSide();
}
#endif
```

Evaluation

- Example case: Correct answer
- Regular chain: Correct answer
- Closed loop: Correct answer

Question 2

C++ . ALGORITHMIC THINKING . QUEUE . PREMIUM

Score: 100%

Time: 8min 53sec

Difficulty: Easy

Description:

Implement the class *Veterinarian* that will be used to track pets waiting in line at a veterinarian's office.

The class *Veterinarian* needs to be **efficient** with respect to time used and contain the following methods:

- *void accept(std::string petName)* - puts the pet at the end of the line.
- *std::string heal()* - removes the pet at the start of line and returns it. If no pets are in line, *std::logic_error("Clinic is empty!")* should be thrown.

For example, the following code snippet should print "Barkley" and then "Mittens":

```
Veterinarian veterinarian;
veterinarian.accept("Barkley");
veterinarian.accept("Mittens");
std::cout << veterinarian.heal() << std::endl; // Should print: Barkley
std::cout << veterinarian.heal() << std::endl; // Should print: Mittens
```

Answer

```
#include <string>
#include <iostream>
#include <stdexcept>
#include <queue>

class Veterinarian
{
public:
    void accept(std::string petName)
    {
        waitinglist.push(petName);
    }

    std::string heal()
    {
        if (waitinglist.empty())
            throw std::logic_error("Clinic is empty!");

        std::string out = waitinglist.front();
        waitinglist.pop();
        return out;
    }
private:

    //First in first out list
    std::queue<std::string> waitinglist;
};

#ifndef RunTests
int main()
{
    Veterinarian veterinarian;
    veterinarian.accept("Barkley");
    veterinarian.accept("Mittens");
    std::cout << veterinarian.heal() << std::endl; // Should print: Barkley
    std::cout << veterinarian.heal() << std::endl; // Should print: Mittens
```

```
}
#endif
```

Evaluation

- Example case: Correct answer
- Small line of pets: Correct answer
- Performance test with many pets: Correct answer

Question 3

C++ . OOP . METHOD OVERRIDING . INHERITANCE . PREMIUM

Score: 50%

Time: 6min 46sec

Difficulty: Easy

Description:

Consider the following C++ code:

```
class MilesToKmConverter
{
public:
    virtual double getMilesToKmFactor()
    {
        return 1.609;
    }

    double milesToKm(double miles)
    {
        return this->getMilesToKmFactor() * miles;
    }
};

class NauticalMilesToKmConverter : public MilesToKmConverter
{
public:
    double getMilesToKmFactor() override
    {
        return 1.852;
    }
};
```

Select all the correct answers.

Answer **Correct answer** | Candidate's selection

- ☐ MilesToKmConverter* converter = new NauticalMilesToKmConverter();
  converter->milesToKm(1);

  will return 1.609.

- ☑ **MilesToKmConverter* converter = new MilesToKmConverter();**
  converter->milesToKm(1);

  will return 1.609.

NauticalMilesToKmConverter* converter = new MilesToKmConverter();
converter->milesToKm(1);

will return 1.852.

NauticalMilesToKmConverter* converter = new NauticalMilesToKmConverter();
converter->milesToKm(1);

will return 1.852.

Question 4

C++ . OOP . ABSTRACT CLASS . PREMIUM . NEW

Score: 100%

Time: 3min 33sec

Difficulty: Easy

Description:

A hospital uses the following class as the basic model for a patient:

```
class Patient
{
public:
    std::string name;

    std::string describe()
    {
        return this->constructDescription();
    }
protected:
    virtual void save() = 0;

    Patient(std::string name)
    {
        this->name = name;
    }

    virtual ~Patient()
    {
    }

    std::string constructDescription()
    {
        return "Patient description: name - " + this->name;
    }
};
```

Select all the correct statements.

Answer **Correct answer** | Candidate's selection

The *constructDescription* method can be called in any class that inherits from the *Patient* class.

Classes inheriting from *Patient* can call *Patient*'s constructor without any arguments.

Any classes that inherit from *Patient* can change the value of the *name* property using *this->name*.

The *Patient* class cannot be instantiated.

- [ ] Classes inheriting from *Patient* must provide their implementation of the *describe* method.
- [x] **A non-abstract class inheriting directly from *Patient* must provide an implementation of the *save* method.**