

Examen de Programmation C++

Durée : 1h30

Aucun document autorisé

Mobiles interdits

Nom :

Prénom :

1 Question 1

- 1. On considère que la classe `std::string` n'existe pas, et on souhaite développer une classe `String` pour représenter les chaînes de caractères. Complétez la classe `String` suivante :

```
class String {
protected:
    char *s; // les caractères de la chaîne
    int lg; // sa longueur
};
```

avec :

1. le **constructeur** qui permet, par exemple, la déclaration `String s1("toto");`
2. le **constructeur** qui permet, par exemple, la déclaration `String s2 = s1;` (attention, on obtient une nouvelle chaîne de caractères)
3. l'**opérateur** qui permet, par exemple, `s2 = s1;` (attention, on obtient une nouvelle chaîne de caractères)
4. l'**opérateur** qui permet d'accéder ou de modifier le i ème ($0 \leq i < lg$) caractère de la chaîne courante. Vous émettez l'exception `IndexException` si l'indice est incorrect.
5. l'**opérateur** qui permet d'écrire `s2 + s2` (concaténation de deux chaînes)
6. la surcharge de l'opérateur `<<`

Note : vous pouvez utiliser les fonctions `strlen`, `strcpy` et `strcat` du langage C (`#include <cstring>`)

```
#pragma once
#include <cstring>
#include <string>
#include <exception>
```

```
class IndexException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Bad index";
    }
};
```

```
class String {
private:
    void dupliquer(const String &s) {
        this->st = new char[this->lg + s.lg+1];
        strcpy(this->st, s.st);
    }

protected:
    char *st; // les caractères de la chaîne
    int lg; // sa longueur
public:
    //
    String(char *s= (char *) "") : st(s), lg(strlen(s)) {}
    // constructeur de copie
    String(const String &s) { dupliquer(s); }
    // surcharge de l'opérateur d'affectation
    const String &operator=(const String &s) { dupliquer(s); }
    // opérateur d'accès au ième caractère
    char &operator[] (const int i) {
        if (i<0 || i>=this->lg) throw IndexException();
        return this->st[i];
    }
    // opérateur de concaténation : renvoie *this+s
    const String operator+(const String &s) const {
        String rs;
        rs.st = new char[rs.lg + this->lg + s.lg+1];
        strcpy(rs.st, this->st);
        rs.st = strcat(rs.st, s.st);
        return rs;
    }
    // surcharge de l'opérateur de l'opérateur d'écriture <
    friend std::ostream&operator<<(std::ostream &f, const String &s) {
        for (int i=0; i<s.lg; i++) f << s.st[i];
        return f;
    }
};
```

2 Question 2

On dispose d'une classe abstraite générique `Liste` et d'une classe d'implémentation générique `ListeChaine` qui implémente à l'aide d'une structure d'éléments dynamiques chaînés.

On souhaite implémenter une *pile* à l'aide d'une liste. On rappelle que les méthodes disponibles sur une liste sont :

```
int longueur() const;
void inserer(const T &x, const int r);
void supprimer(const int r);
const T &ieme(int r) const;
```

- 2. Écrivez en C++ la classe abstraite générique `Pile` avec les quatre méthodes abstraites *estVide*, *empiler*, *dépiler*, et *sommet*.

```
#pragma once
template <typename T>
class Pile {
public:
    virtual bool estVide() const =0;
    virtual void empiler(const T& x) =0;
    virtual void depiler() =0;
    virtual const T& sommet() const =0;
};
```

- 3. En vous servant de la classe `ListeChaine`, écrivez en C++ la classe générique `PileChaine` qui implémente une `Pile` à l'aide d'une liste chaînée. Vous programmez le(s) constructeur(s) et les quatre méthodes de manipulation d'une pile. Vous émettez l'exception `PileVideException` si nécessaire.

```
#pragma once
#include "ListeChaine.hpp"
#include "Pile.hpp"
#include <exception>

class PileVideException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Pile vide";
    }
};

template <typename T>
class PileChaine : public Pile<T> {
private:
    ListeChaine<T> p;
public:
    PileChaine() {}

    // méthodes de base de manipulation de la pile
    bool estVide() const override {
        return p.longueur()==0;
    }

    void empiler(const T &x) override {
        p.insérer(x, 1);
    }

    void depiler() override {
        if (this->estVide()) throw PileVideException();
        p.supprimer(1);
    }

    const T &sommet() const override {
        if (this->estVide()) throw PileVideException();
        return p.ième(1);
    }
};
```

```
}
};
```

- 4. Écrivez en C++ la fonction `main` qui teste votre classe `PileChaine`, et en particulier attrape une exception.

```
int main() {
    PileChaine<string> pi;
    pi.empiler("toto");
    pi.empiler("titi");
    std::cout << pi << std::endl;
    std::cout << pi.sommet() << std::endl;
    pi.depiler();
    pi.depiler();
    try {
        std::cout << pi.sommet() << std::endl;
    }
    catch (PileVideException &e) {
        std::cerr << "La pile est vide !!!" << std::endl;
    }
    return EXIT_SUCCESS;
}
```
