Université de Nice-Sophia Antipolis — POLYTECH
ELEC4 — 2022–2023

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

Une *file d'attente* est une structure de données linéaire dans laquelle on entre par une extrémité et on en sort par l'autre. On définit les 4 opérations suivantes :

- `estVide` qui teste si la file est vide ou pas ;
- `premier` qui renvoie le premier élément de la file ;
- `defiler` qui enlève le premier élément de la file ;
- `enfiler` qui ajoute un élément à la fin de la file.

▶ 1. Écrivez la classe abstraite générique `File` avec les 4 opérations précédentes, la méthode `toString` et l'opérateur `<<` pour écrire dans un `std::ostream`.

```
template <typename T>
class File {
public:
  /*
   * Rôle : renvoie true si la file courante est vide,
   *          et false sinon
   */
  virtual bool estVide() const =0;
  /*
   * Rôle : ajoute l'élément x à la fin la file courante
   */
  virtual void enfiler(const T &x) =0;
  /*
   * Rôle : supprime le premier élément de la file courante
   */
  virtual void defiler() =0;
  /*
   * Rôle : renvoie le premier élément de la file courante
   */
  virtual const T &premier() const =0;
  /*
   * Rôle : renvoie la représentation de l'objet courant sous forme d'une string
   */
  virtual const std::string toString() const =0;

  // rôle : écrit la contenu de la file p sur f
  friend std::ostream&operator<<(std::ostream &f, const File<T>& fi) {
    return f << fi.toString();
  }
};
```

▶ 2. Écrivez la classe `FileVideException` qui dérive de `std::exception`. Vous redéfinirez uniquement la méthode `what`.


```
class FileVideException : public std::exception {
public:
  // redéfinition de what
  virtual const char* what() const noexcept override {
    return "File vide Exception";
  }
};
```

▶ 3. Écrivez la classe générique `FileVecteur` qui implémente une file d'attente par héritage de la classe `File`. Les éléments de `FileVecteur` sont conservés dans une variable membre privée de type `std:vector` (voir annexe jointe). Les méthodes `premier` et `defiler` émettront l'exception `FileVideException`.

```
template <typename T>
class FileVecteur : public File<T> {
protected:
  std::vector<T> lesElements; // vecteur générique d'éléments de type T
public:
  virtual bool estVide() const override {
    return this->lesElements.empty();
  }
  virtual const T &premier() const override {
    if (this->estVide())
      throw FileVideException();
    // file non vide
    return this->lesElements.front();
  }
  virtual void defiler() override {
    if (this->estVide())
      throw FileVideException();
    // file non vide
    this->lesElements.erase(lesElements.begin());
  }
  virtual void enfiler(const T &x) override {
    this->lesElements.push_back(x);
  }
  virtual const std::string toString() const override {
    std::ostringstream s;
    s << "out <- [ ";
    if (!this->estVide()) {
      s << this->lesElements[0] << " | ";
      for (unsigned int i=1; i<this->lesElements.size(); i++)
        s << this->lesElements[i] << " ";
    }
    s << "] <- in";
    return s.str();
  }
};
```

Les véhicules qui circulent en France sur les autoroutes sont classés en 5 catégories :

— classe 1 : véhicules légers (*e.g.* cabriolet) ;
— classe 2 : véhicules intermédiaires (*e.g.* camping-car) ;
— classe 3 : véhicules lourds à 2 essieux (*e.g.* autocar) ;
— classe 4 : véhicules lourds à +3 essieux (*e.g.* camion 38T) ;
— classe 5 : véhicules motos, side-cars...

▶ 4. Écrivez la classe `Vehicule` avec une variable membre protégée `classe` et une méthode `getClasse`.

```
class Vehicule {
protected:
  int classe;
public:
  // initialise le véhicule courant de classe c
  Vehicule(const int c) : classe(c) {
    assert(c>=1 && c<=5);
  }
  // renvoie la classe du véhicule courant
  int getClasse() const {
    return this->classe;
  }
};
```

.....

▶ 5. Écrivez les classes `Cabriolet` et `Moto` avec leur constructeur. Quelle relation existe entre ces 2 classes et la classe `Vehicule` ?

Les classes `Cabriolet` et `Moto` héritent de `Vehicule`. Ce sont des véhicules particuliers.

```
class Cabriolet : public Vehicule {
public:
  Cabriolet() : Vehicule(1) {}
};

class Moto : public Vehicule {
public:
  Moto() : Vehicule(5) {}
};
```

.....

On représente un péage d'autoroute par la classe `Peage`. Cette classe comporte :
— une variable privée `tarifs`, un tableau initialisé avec les tarifs pour les 5 catégories de véhicule ;
— une méthode `getTarif` qui renvoie le tarif d'un véhicule en fonction de sa classe. Cette fonction a en paramètre un pointeur sur véhicule. Attention : ne faire aucun test !
— une méthode `encaisser` qui prend en paramètre un pointeur sur une `File` de pointeurs sur `Vehicule` et qui renvoie la somme totale des tarifs payés par les véhicules de la file.

▶ 6. Écrivez la classe `Peage`.

```
class Peage {
private:
  double tarifs[6] = { -1, 7, 9, 10, 15, 3 }; // note :tarifs[0] invalide
```



```
public:

  double getTarif(const Vehicule *v) const {
    return this->tarifs[v->getClasse()];
  }

  double encaisser(File<Vehicule*> *f) const {
    double somme = 0.0;
    while (!f->estVide()) {
      somme+= this->getTarif(f->premier());
      f->defiler();
    }
    return somme;
  }
};
```

.....

▶ 7. Écrivez la méthode `main` qui :
— déclare une variable `fv` de type pointeur sur une `File` de pointeurs sur `Vehicule` ;
— enfile dans `fv` trois `Cabriolet` et deux `Moto` ;
— déclare une variable `p` de type `Peage` ;
— affiche sur la sortie standard l'encaissement total des tarifs des véhicules de la file `fv`.

```
int main() {

  File<Vehicule*> *fv = new FileVecteur<Vehicule*>();

  fv->enfiler(new Moto());
  fv->enfiler(new Cabriolet());
  fv->enfiler(new Cabriolet());
  fv->enfiler(new Cabriolet());
  fv->enfiler(new Moto());
  //
  Peage p;
  //
  std::cout << p.encaisser(fv) << std::endl;
  //
  return EXIT_SUCCESS;
}
```

.....

▶ 8. Expliquez, de façon claire et synthétique, la notion de *liaison dynamique* des langages à objets tels que C++.

Cf. cours.

.....

Annexe

Member functions

(constructor)	constructs the `vector` (public member function)
(destructor)	destructs the `vector` (public member function)
`operator=`	assigns values to the container (public member function)
`assign`	assigns values to the container (public member function)
`get_allocator`	returns the associated allocator (public member function)
Element access	
`at`	access specified element with bounds checking (public member function)
`operator[]`	access specified element (public member function)
`front`	access the first element (public member function)
`back`	access the last element (public member function)
`data`	direct access to the underlying array (public member function)
Iterators	
`begin` **`cbegin`** (C++11)	returns an iterator to the beginning (public member function)
`end` **`cend`** (C++11)	returns an iterator to the end (public member function)
`rbegin` **`crbegin`** (C++11)	returns a reverse iterator to the beginning (public member function)
`rend` **`crend`** (C++11)	returns a reverse iterator to the end (public member function)
Capacity	
`empty`	checks whether the container is empty (public member function)
`size`	returns the number of elements (public member function)
`max_size`	returns the maximum possible number of elements (public member function)
`reserve`	reserves storage (public member function)
`capacity`	returns the number of elements that can be held in currently allocated storage (public member function)
`shrink_to_fit` (C++11)	reduces memory usage by freeing unused memory (public member function)
Modifiers	
`clear`	clears the contents (public member function)
`insert`	inserts elements (public member function)
`emplace` (C++11)	constructs element in-place (public member function)
`erase`	erases elements (public member function)
`push_back`	adds an element to the end (public member function)
`emplace_back` (C++11)	constructs an element in-place at the end (public member function)
`pop_back`	removes the last element (public member function)
`resize`	changes the number of elements stored (public member function)
`swap`	swaps the contents (public member function)