POLYTECH – ELEC4 2024–2025

Contrôle de COO

Durée : 1h00 Aucun document autorisé

Nom : Prénom :

▶ 1. Expliquez de façon claire et synthétique la notion *générale* d'itérateur et celle, particulière, de C++.

▶ 2. Récrivez la fonction `find_if` qui recherche dans un conteneur un élément qui vérifie un prédicat booléen. La fonction recherche entre deux itérateurs, et renvoie l'itérateur qui désigne le premier élément qui vérifie le prédicat. Si aucun élément ne vérifie le prédicat, la fonction renvoie l'itérateur de fin. L'en-tête de cette fonction est le suivant :

```
auto find_if(auto itDébut, auto itFin, auto prédicat)
```

▶ 3. Avec votre fonction `find_if`, écrivez le fragment de code qui recheche dans tout un vecteur `v` de `std::string` la première chaîne de caractères de taille paire. Si elle est trouvée, vous l'écrirez sur la sortie standard. Vous écrirez le prédicat booléen sous forme d'une *fonction anonyme*.

On désire écrire une application qui compte le nombre d'occurrences des mots contenus dans un fichier de texte. Le programme écrit sur la sortie standard la liste des mots lus en ordre alphabétique avec, pour chaque mot, son nombre d'occurrences.

Pour mémoriser les mots et leur nombre d'occurrences, on définit la classe `Dico` qui héritera de la classe générique `std::map`. Dans cette classe, vous écrirez la méthode `insérer` pour ajouter un mot dans le dictionnaire, la méthode `toString` et l'opérateur `<<` pour écrire tous les mots du dictionnaire dans l'ordre l'alphabétique avec leurs nombres d'occurrences.

▶ 4. Écrivez la classe `Dico`.

▶ 5. On veut écrire une application qui permet à son utilisateur de trouver le chemin qui relie deux lieux (représentés par des `std::string`) dans une ville, qu'il soit à pied, en bus ou en voiture. L'application suivra le patron de conception `stratégie`. Elle est formée de 5 classes : `stratégie`, `enVoiture`, `ÀPied`, `EnBus`, et `Navigateur`. Le programme principal est le suivant :

```
void gps(const stratégie &s, const std::string &from, const std::string &to) {
  Navigateur(s).trouverLeChemin(from, to);
}

int main() {
  ÀPied àPied;
  EnVoiture voiture;
  EnBus bus;

  gps(àPied, "Maison", "Piscine");
  gps(bus, "Maison", "Polytech'Sophia");
  gps(voiture, "Musée", "12 Rue St Paul");

  return EXIT_SUCCESS;
};
```

Il produit sur la sortie standard le résultat suivant :

```
à pied de Maison à Piscine, il faut passer par ...
en bus de Maison à Polytech'Sophia, il faut passer par ...
en voiture de Musée à 12 Rue St Paul, il faut passer par ...
```

On donne la classe `stratégie` :

```
class  stratégie {
public:
  virtual void trouverLeChemin(const std::string &a, const std::string &b)
  const =0;
};
```

▶ 6. Dessinez le diagramme de classes en UML, et complétez les classes données ci-dessous :

```
/**
 *
 */
class  EnVoiture ...




};

/**
 *
 */
class  ÀPied ...




};

/**
 *
 */
class  EnBus ...




};

/**
 *
 */
class Navigateur ...




};
```