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

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

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

Cf. cours.

.....

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

Cf. cours

.....

▶ 3. Déclarez une variable `l` (elle) de type liste d'entiers.

```
std::list<int> l;
```

.....

▶ 4. Ajoutez en tête de liste les valeurs 4, puis 9.

```
l.push_front(4);
l.push_front(9);
```

.....

▶ 5. Ajoutez en fin de liste la valeur 0.

```
l.push_back(0);
```

.....

▶ 6. Déclarez un itérateur associé à la liste `l`, et insérez la valeur -1 en 2ème position. Vous modifierez l'itérateur pour qu'il désigne la valeur que vous venez d'insérer.

```
std::list<int>::iterator it=l.begin();
l.insert(++it, -1);
```

.....

▶ 7. Écrivez sur la sortie standard la valeur en deuxième position dans la liste.

```
std::cout << *it << std::endl;
```

.....

▶ 8. Supprimez la dernière valeur de la liste.

```
l.pop_back();
```

.....

▶ 9. Supprimez toutes les valeurs du début de la liste jusqu'à la position donnée par l'itérateur.

```
l.erase(l.begin(), ++it);
```

.....

▶ 10. Écrivez la procédure *générique* `ecrireListe` qui écrit sur la sortie standard les éléments d'une liste passée en paramètre.

```
template<typename T>
void ecrireListe(const std::list<T> &l) {
  std::cout << "< ";
  for (const T &x : l)
    std::cout << x << " ";
  std::cout << ">" << std::endl;
}
```

.....

▶ 11. Écrivez la procédure générique `echanger` qui échange deux éléments d'une liste générique repérés chacun par un itérateur. La liste et les deux itérateurs sont passés en paramètre de la procédure. Cette procédure ne crée, ni supprime d'élément.

```
template<typename T>
void echanger(std::list<T> &l, const auto &pos1, const auto &pos2) {
  T aux = *pos1;
  *pos1 = *pos2;
  *pos2 = aux;
}
```

▶ 12. À l'aide de la procédure `echanger` précédente, écrivez la procédure générique `inverser` qui inverse les éléments d'une liste générique passée en paramètre. Le premier élément devient le dernier, le second l'avant dernier, *etc.* Cette procédure ne crée, ni supprime d'élément.

```
template<typename T>
void inverser(std::list<T> &l) {
  auto debut =  l.begin();
  auto fin =  l.end();
  const int milieu = l.size()/2; // pas d'opérateur < sur les itérateurs !
  fin--;
  for (int i=0; i<milieu; i++)
    echanger(l, debut++, fin--);
}
```

▶ 13. À l'aide des procédures précédentes, écrivez la fonction `main` qui déclare une liste initialisée avec 5 entiers (**int**), qui l'écrit sur la sortie standard, l'inverse, et l'écrit à nouveau.

```
int main () {
  std::list<int> l = { 1, 2, 3, 4, 5 };

  ecrireListe(l);
  inverser(l);
  ecrireListe(l);

  return EXIT_SUCCESS;
}
```

▶ 14. Écrivez en C++ programme qui compte le nombre d'occurrences des mots contenus dans un fichier de texte. Le programme affichera la liste des mots lus en ordre alphabétique avec, sur chaque ligne et pour chaque mot, son nombre d'occurrences.

Pour mémoriser les mots et leur nombre d'occurrences, vous définirez la classe `Dico`, **héritière** de la classe générique `std::map`. Dans cette classe, vous écrirez la méthode `inserer` 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.

On rappelle que la méthode `find` recherche un élément dans une `std::map` à partir de sa clé passée en paramètre. Cette méthode renvoie la position (un itérateur) de l'élément dans la table, ou la position de fin si l'élément n'est pas présent dans la table. On rappelle aussi que tous les couples, clé/valeurs, contenues dans une `std::map` sont mémorisées dans des `std::pair`.

```
class Dico : private std::map<std::string, int> {
public:
  // Rôle : insère dans la table t le mot m
  void inserer(std::string m) {
    (*this)[m] = this->find(m)==this->end() ? 1 : (*this)[m]+1;
  }
  // avec c++17 !
  std::string toString() const {
    std::ostringstream s;
    for (const auto& [key, value] : *this)
     s << key << " : " << value << "\n";
    return s.str();
  }

  friend std::ostream &operator<<(std::ostream &f, const Dico& d) {
    return f << d.toString();
  }
};

int main (int argc, char *argv[]) {
  if (argc!=2) {
    std::cerr <<  "Usage: xref file" << std::endl;
    std::exit(EXIT_FAILURE);
  }
  // ouverture du fichier d'entrée
  std::ifstream ios(argv[1]);
  if (!ios.is_open()) {
    perror(argv[1]);
    std::exit(EXIT_FAILURE);
  }
  // déclaration de table pour mémoriser les mots
  // et leur nombre d'apparitions
  Dico table;
  // lire les mots du fichier
  std::string s;
  while  (ios >> s)
    table.inserer(s);
  //
  ios.close();
  // afficher la table
  std::cout << table;

  return EXIT_SUCCESS;
}
```

1 Annexe

1.1 std::list

Element access	
front	access the first element (public member function)
back	access the last element (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)
Modifiers	
clear	clears the contents (public member function)
insert	inserts elements (public member function)
insert_range (C++23)	inserts a range of 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)
append_range (C++23)	adds a range of elements to the end (public member function)
pop_back	removes the last element (public member function)
push_front	inserts an element to the beginning (public member function)
emplace_front (C++11)	constructs an element in-place at the beginning (public member function)
prepend_range (C++23)	adds a range of elements to the beginning (public member function)
pop_front	removes the first element (public member function)
resize	changes the number of elements stored (public member function)
swap	swaps the contents (public member function)
Operations	
merge	merges two sorted lists (public member function)
splice	moves elements from another list (public member function)
remove remove_if	removes elements satisfying specific criteria (public member function)
reverse	reverses the order of the elements (public member function)
unique	removes consecutive duplicate elements (public member function)
sort	sorts the elements (public member function)

1.2 std::map

Element access	
at	access specified element with bounds checking (public member function)
operator[]	access or insert specified element (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)
Modifiers	
clear	clears the contents (public member function)
insert	inserts elements or nodes (since C++17) (public member function)
insert_range (C++23)	inserts a range of elements (public member function)
insert_or_assign (C++17)	inserts an element or assigns to the current element if the key already exists (public member function)
emplace (C++11)	constructs element in-place (public member function)
emplace_hint (C++11)	constructs elements in-place using a hint (public member function)
try_emplace (C++17)	inserts in-place if the key does not exist, does nothing if the key exists (public member function)
erase	erases elements (public member function)
swap	swaps the contents (public member function)
extract (C++17)	extracts nodes from the container (public member function)
merge (C++17)	splices nodes from another container (public member function)
Lookup	
count	returns the number of elements matching specific key (public member function)
find	finds element with specific key (public member function)
contains (C++20)	checks if the container contains element with specific key (public member function)
equal_range	returns range of elements matching a specific key (public member function)
lower_bound	returns an iterator to the first element *not less* than the given key (public member function)
upper_bound	returns an iterator to the first element *greater* than the given key (public member function)
Observers	

1.3 std::pair

std::pair

Defined in header <utility>

```
template<
    class T1,
    class T2
> struct pair;
```

std::pair is a class template that provides a way to store two heterogeneous objects as a single unit. A pair is a specific case of a std::tuple with two elements.

If neither T1 nor T2 is a possibly cv-qualified class type with non-trivial destructor, or array thereof, the destructor of pair is trivial.

Template parameters

T1, T2 - the types of the elements that the pair stores.

Member types

Member type	Definition
first_type	T1
second_type	T2

Member objects

Member name	Type
first	T1
second	T2

Member functions

(constructor)	constructs new pair (public member function)
operator=	assigns the contents (public member function)
swap (C++11)	swaps the contents (public member function)