

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

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

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

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

- ▶ 8. Supprimez la dernière valeur de la liste.

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

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

- ▶ 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.

- ▶ 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.

- ▶ 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.

1 Annexe

1.1 `std::list`

Element access	
<code>front</code>	access the first element (public member function)
<code>back</code>	access the last element (public member function)
Iterators	
<code>begin</code> <code>cbegin</code> (C++11)	returns an iterator to the beginning (public member function)
<code>end</code> <code>cend</code> (C++11)	returns an iterator to the end (public member function)
<code>rbegin</code> <code>crbegin</code> (C++11)	returns a reverse iterator to the beginning (public member function)
<code>rend</code> <code>crend</code> (C++11)	returns a reverse iterator to the end (public member function)
Capacity	
<code>empty</code>	checks whether the container is empty (public member function)
<code>size</code>	returns the number of elements (public member function)
<code>max_size</code>	returns the maximum possible number of elements (public member function)
Modifiers	
<code>clear</code>	clears the contents (public member function)
<code>insert</code>	inserts elements (public member function)
<code>insert_range</code> (C++23)	inserts a range of elements (public member function)
<code>emplace</code> (C++11)	constructs element in-place (public member function)
<code>erase</code>	erases elements (public member function)
<code>push_back</code>	adds an element to the end (public member function)
<code>emplace_back</code> (C++11)	constructs an element in-place at the end (public member function)
<code>append_range</code> (C++23)	adds a range of elements to the end (public member function)
<code>pop_back</code>	removes the last element (public member function)
<code>push_front</code>	inserts an element to the beginning (public member function)
<code>emplace_front</code> (C++11)	constructs an element in-place at the beginning (public member function)
<code>prepend_range</code> (C++23)	adds a range of elements to the beginning (public member function)
<code>pop_front</code>	removes the first element (public member function)
<code>resize</code>	changes the number of elements stored (public member function)
<code>swap</code>	swaps the contents (public member function)
Operations	
<code>merge</code>	merges two sorted lists (public member function)
<code>splice</code>	moves elements from another list (public member function)
<code>remove</code> <code>remove_if</code>	removes elements satisfying specific criteria (public member function)
<code>reverse</code>	reverses the order of the elements (public member function)
<code>unique</code>	removes consecutive duplicate elements (public member function)
<code>sort</code>	sorts the elements (public member function)

1.2 std::map

Element access	
<code>at</code>	access specified element with bounds checking (public member function)
<code>operator[]</code>	access or insert specified element (public member function)
Iterators	
<code>begin</code> <code>cbegin</code> (C++11)	returns an iterator to the beginning (public member function)
<code>end</code> <code>cend</code> (C++11)	returns an iterator to the end (public member function)
<code>rbegin</code> <code>crbegin</code> (C++11)	returns a reverse iterator to the beginning (public member function)
<code>rend</code> <code>crend</code> (C++11)	returns a reverse iterator to the end (public member function)
Capacity	
<code>empty</code>	checks whether the container is empty (public member function)
<code>size</code>	returns the number of elements (public member function)
<code>max_size</code>	returns the maximum possible number of elements (public member function)
Modifiers	
<code>clear</code>	clears the contents (public member function)
<code>insert</code>	inserts elements or nodes (since C++17) (public member function)
<code>insert_range</code> (C++23)	inserts a range of elements (public member function)
<code>insert_or_assign</code> (C++17)	inserts an element or assigns to the current element if the key already exists (public member function)
<code>emplace</code> (C++11)	constructs element in-place (public member function)
<code>emplace_hint</code> (C++11)	constructs elements in-place using a hint (public member function)
<code>try_emplace</code> (C++17)	inserts in-place if the key does not exist, does nothing if the key exists (public member function)
<code>erase</code>	erases elements (public member function)
<code>swap</code>	swaps the contents (public member function)
<code>extract</code> (C++17)	extracts nodes from the container (public member function)
<code>merge</code> (C++17)	splices nodes from another container (public member function)
Lookup	
<code>count</code>	returns the number of elements matching specific key (public member function)
<code>find</code>	finds element with specific key (public member function)
<code>contains</code> (C++20)	checks if the container contains element with specific key (public member function)
<code>equal_range</code>	returns range of elements matching a specific key (public member function)
<code>lower_bound</code>	returns an iterator to the first element <i>not less</i> than the given key (public member function)
<code>upper_bound</code>	returns an iterator to the first element <i>greater</i> 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
<code>first_type</code>	T1
<code>second_type</code>	T2

Member objects

Member name	Type
<code>first</code>	T1
<code>second</code>	T2

Member functions

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