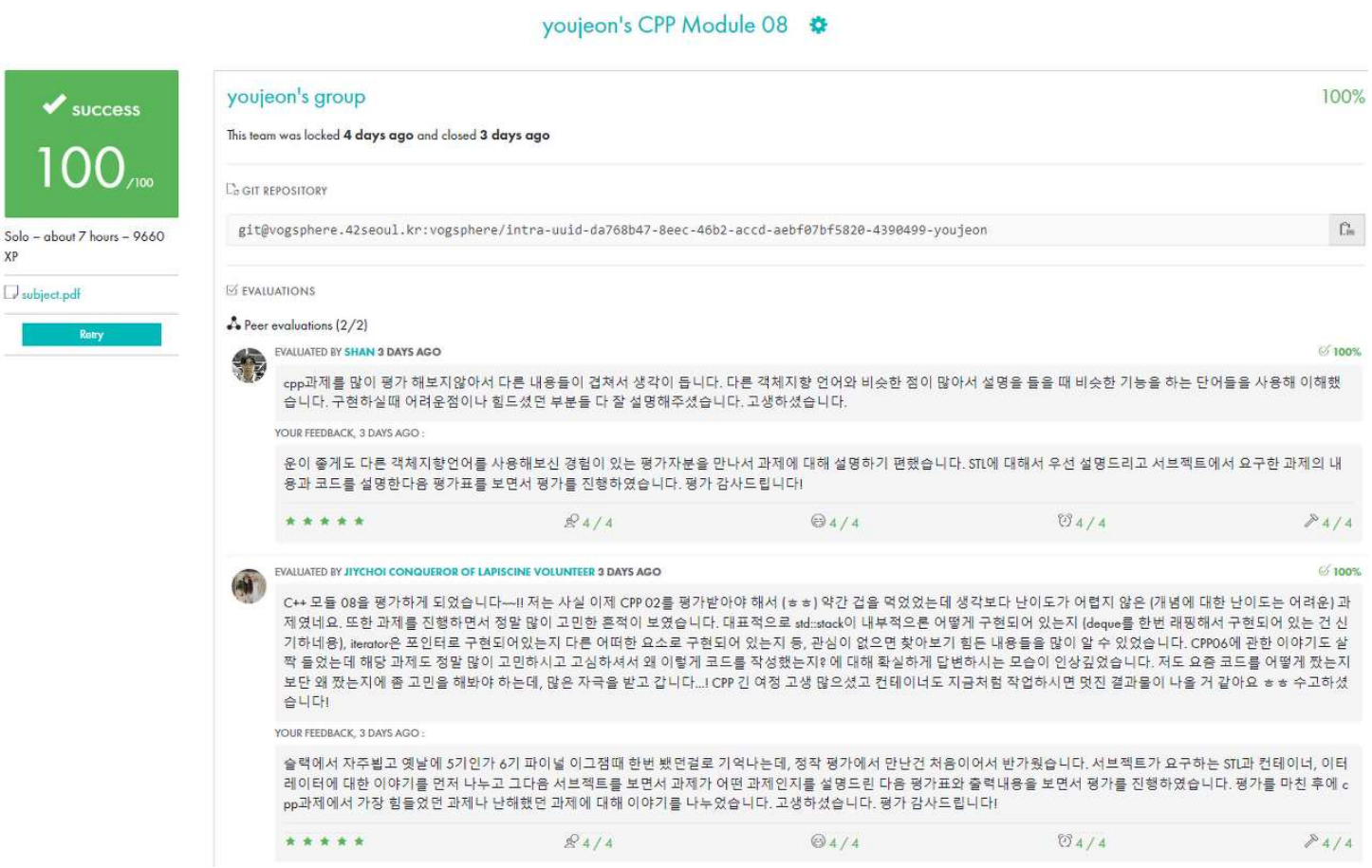
programación

[42Seúl] Módulo CPP 08 - STL

Aggrodonk 2022. 9. 11. 22:48

Introducción



Por supuesto, se puede implementar sin STL, pero como el propósito de la tarea es usarlo, intent aré usarlo tanto como sea posible.

La plantilla se puede definir en un archivo de encabezado o en un archivo tpp. En todos los caso s, tpp es opcional y hpp es obligatorio.

> La Biblioteca de plantillas estándar (STL) es un conjunto de plantillas que representan contenedores, iteradores, objetos de función y algoritmos... . Un contenedor es una unidad estructural que puede almacenar múltiples valores, como una matriz... Los algoritmos son métodos que se utilizan para realizar operaciones especiales, como ordenar una matriz o buscar un valor específico en una lista. Los iteradores son objetos que le ayudan a mover la posición dentro de un contenedor, del mismo modo que utiliza un puntero para mover la posición dentro de una matriz. En otras palabras, un iterador es una generalización de un puntero. - [Conceptos básicos de C++ Plus 1241p. Seongandang]

https://cplusplus.com/reference/stl/

	Contenedores: referencia de C++ cplusplus.com

ex00

Cree una función easyfind que tome dos argumentos (el primer argumento es T y el segundo arg umento es un número entero).

Suponiendo que T es un contenedor de números enteros, encuentre la primera aparición del seg undo argumento en el primer argumento.

Si no se encuentra, puede generar una excepción o manejarla devolviendo un valor de error de s u elección. Es una buena idea inspirarse en cómo lo manejan los contenedores estándar.

Por supuesto, cree una declaración principal, implemente la prueba y envíela.

Los contenedores asociativos no necesitan ser procesados

```
template <typename T>
typename T::iterator easyfind(T& container, int value)
{
        typename T::iterator iter;

        iter = std::find(container.begin(), container.end(), value);
        if (iter == container.end())
        {
                throw std::runtime_error("value is not in this container");
        }
        return iter;
}
```

07 Por último, la tarea es hacer que la función de plantilla aplicada a la matriz sea utilizable en el contenedor.

Sentí curiosidad y lo probé porque escuché que los contenedores relacionados no necesitan ser procesados. No hubo ningún problema importante con el conjunto, la entrada y la salida eran lig eramente diferentes, pero en el caso del mapa, había muchas cosas que sí. procesarse por separ ado, así que lo entendí como una tarea para hacerlo simple.

Si observamos cómo el contenedor estándar maneja la búsqueda, se maneja especificando iter c omo contenedor.end(). end es la siguiente dirección al final de los datos del contenedor y, para u n uso conveniente en bucles for, etc., apunta a la siguiente dirección en lugar de a la última direc ción. Si es así, no creo que signifique volver a crear una. función ya existente (No reinventar la rue da) Creé una función que arroja una excepción.

Por supuesto, find requiere especificar el inicio y el final del iterador, y si easyfind se crea pensan do que es una función para encontrar todo el contenedor más fácilmente, creo que es un área de elección que se puede procesar igual que el regreso del hallazgo.

```
int main(void)
{
        std::vector<int> v;
        std::deque<int> d;
        std::list<int> l;

        for (int i = 0; i < 10; i++)
        {
                v.push_back(i);
                d.push_back(i);
                l.push_back(i);
        }
```

```
	try
	{
		std::cout << *(easyfind(v, 5)) << " is at " << std::distance(v.begin(), easyfi
nd(v, 5)) << std::endl;
	}
	catch(const std::exception& e)
	{
		std::cerr << e.what() << '\n';
	}
	try
	{
		std::cout << *(easyfind(d, 0)) << " is at " << std::distance(d.begin(), easyfi
nd(d, 5)) << std::endl;
	}
	catch(const std::exception& e)
	{
		std::cerr << e.what() << '\n';
	}
	try
	{
		std::cout << *(easyfind(l, 10)) << " is at " << std::distance(l.begin(), easyf
ind(l, 5)) << std::endl;
	}
	catch(const std::exception& e)
	{
		std::cerr << e.what() << '\n';
	}

	return (0);
}
```

Cuando coloca un número entero del 0 al 9 en cada contenedor y busca el valor, se genera la dist ancia desde el inicio. Cuando vi que se agregó * para generar el valor, lo primero que pensé fue q ue el iterador era un puntero... Eso pensé, pero luego descubrí que la clase del objeto estaba esp ecificada para generar un valor cuando se ingresa *, como un puntero.

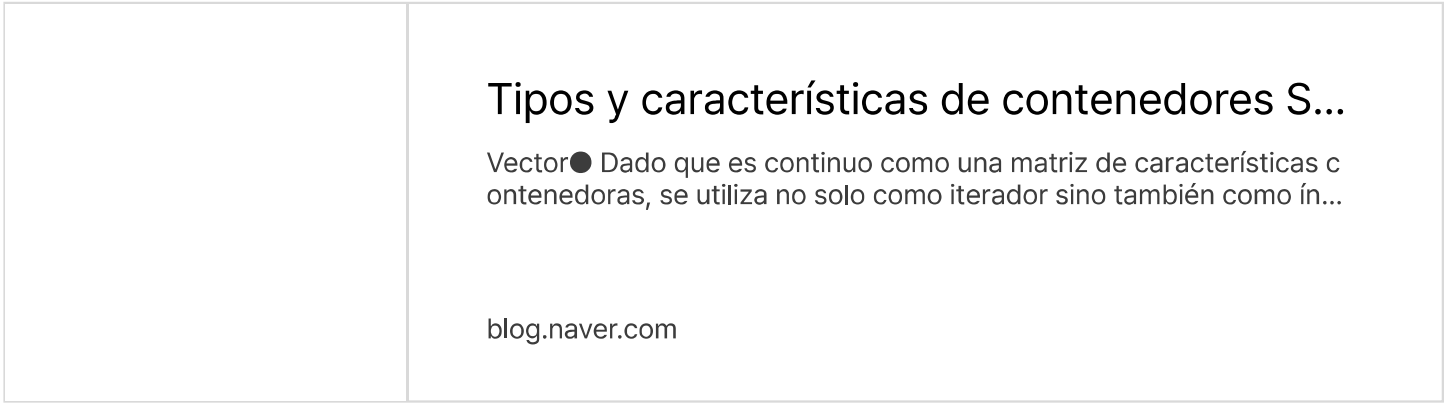
> Los iteradores son una generalización de punteros. De hecho, los iteradores también pueden ser punteros. Alternativamente, puede ser un objeto para el cual se definen operaciones similares a punteros, como referencia de contenido e incremento. - [Conceptos básicos de C++ Plus 1244p. Seongandang]

Al principio, para comprobar en qué índice estaba, lo implementé restando el inicio del iterador a ctual sin usar la distancia, pero luego descubrí que había algunos contenedores que funcionaban como quería y otros que no. Así que busqué de nuevo, encontré la función y la implementé usán dola.

```cpp
template <class Iterator>
class reverse_iterator
    : public iterator<typename iterator_traits<Iterator>::iterator_category, // until C++17
                      typename iterator_traits<Iterator>::value_type,
                      typename iterator_traits<Iterator>::difference_type,
                      typename iterator_traits<Iterator>::pointer,
                      typename iterator_traits<Iterator>::reference>
{
protected:
    Iterator current;
public:
    typedef Iterator                                                iterator_type;
    typedef typename iterator_traits<Iterator>::difference_type difference_type;
    typedef typename iterator_traits<Iterator>::reference       reference;
    typedef typename iterator_traits<Iterator>::pointer         pointer;

    constexpr reverse_iterator();
    constexpr explicit reverse_iterator(Iterator x);
    template <class U> constexpr reverse_iterator(const reverse_iterator<U>& u);
    template <class U> constexpr reverse_iterator& operator=(const reverse_iterator<U>& u);
    constexpr Iterator base() const;
    constexpr reference operator*() const;
    constexpr pointer   operator->() const;
    constexpr reverse_iterator& operator++();
    constexpr reverse_iterator  operator++(int);
    constexpr reverse_iterator& operator--();
    constexpr reverse_iterator  operator--(int);
    constexpr reverse_iterator  operator+ (difference_type n) const;
    constexpr reverse_iterator& operator+=(difference_type n);
    constexpr reverse_iterator  operator- (difference_type n) const;
    constexpr reverse_iterator& operator-=(difference_type n);
    constexpr reference         operator[](difference_type n) const;
};
```

https://blog.naver.com/ya3344/221360287260



https://m.blog.naver.com/ktm0122/20167641378

[C++ efectivo] Artículo 42. Entendamos a...

Al declarar un parámetro de tipo de una plantilla, el significado de cla se y nombre de tipo es exactamente el mismo. Pero siempre será cl...

blog.naver.com

https://eehoeskrap.tistory.com/263

[C++] Iterador

Iterador de C++ La biblioteca de C++ proporciona un iterador que le permite acceder a estructuras de datos a la manera de la biblioteca...

eehoeskrap.tistory.com

https://lecor.tistory.com/77

[C++] Imprimir mapa

Descripción general de STL Debe usarse con std:: porque está defini do en el espacio de nombres std, una biblioteca que proporciona la...

lecor.tistory.com

https://www.techiedelight.com/ko/find-index-of-an-element-in-array-cpp/

C++ Buscar índice de elemento en matriz

Esta publicación explica cómo encontrar el índice en el que aparece por primera vez un elemento en una matriz de C++. 1. El uso de std...

www.techiedelight.com

ex01

Cree una clase Span que pueda tomar un número N y almacenar N enteros. N es una variable int sin signo y es el único parámetro.

Esta clase tiene una función addNumber que agrega un número a Span. Lanza una excepción si i ngresas un número que ya existe.

ShortestSpan y longSpan se implementan para encontrar y devolver el rango más corto (es decir, la distancia) y el rango más largo entre los números almacenados. Si no hay ningún número alma cenado o solo hay uno, no se puede encontrar el rango, por lo que se debe generar una excepció n.

Se proporciona la declaración principal, pero debe probarse utilizando al menos 10000 números.

Por último, sería bueno poder llenar el Span utilizando varios iteradores. AddNumber de miles d e números es exasperante, así que implemente una función que le permita agregar muchos núm eros a la vez como función miembro.

Si no está seguro, debería mirar el contenedor y algunas funciones miembro deberían usar iterad ores.

```
int main()
{
Span sp = Span(5);
sp.addNumber(6);
sp.addNumber(3);
sp.addNumber(17);
sp.addNumber(9);
sp.addNumber(11);
std::cout << sp.shortestSpan() << std::endl;
std::cout << sp.longestSpan() << std::endl;
return 0;
}
```

```
$> ./ex01
2
14
$>
```

```
std::size_t Span::longestSpan() const
{
        if (v.size() <= 2)
```

```
        {
                throw std::logic_error("vector size is not over 2");
        }
        return (*std::max_element(v.begin(), v.end()) - *std::min_element(v.begin(), v.end())
);
}

std::size_t Span::shortestSpan() const
{
        if (v.size() <= 2)
        {
                throw std::logic_error("vector size is not over 2");
        }
        long ret = LONG_MAX;
        int prev;
        std::vector<int> tmp = v;

        std::sort(tmp.begin(), tmp.end());
        for (std::vector<int>::iterator iter = tmp.begin(); iter != tmp.end(); iter++)
        {
                if (iter == tmp.begin())
                {
                        prev = *iter;
                }
                else
                {
                        if (ret > *iter - prev)
                        {
                                ret = *iter - prev;
                        }
                        prev = *iter;
                }
        }
        return static_cast<std::size_t>(ret);
}
```

Fue una de las pocas tareas que le tomó más de un día a un ex.

Resolver algo fácil es fácil y resolver algo difícil es difícil, pero el proceso de descubrir qué era fáci l fue más complicado de lo que pensaba. Si hubiera seguido mi primer pensamiento, habría sido más rápido, pero creo que habría aprendido algo menos.

Los demás contenidos de la clase no son tan difíciles, así que me los saltaré, y longSpan tampoco es tan difícil. Si utiliza la función max_element y la función min_element para restar el valor más pequeño del valor más grande en el vector, ese es el intervalo más largo.

shortSpan es un poco complicado y, después de considerar varios métodos, elegí el método más limpio en términos de código para la evaluación. Después de copiar el vector, ordénelo y enumér elo del valor más pequeño al más grande. Luego, observaron el valor anterior y el siguiente uno p or uno, lo compararon con el ret existente y guardaron el más pequeño.

https://codingdog.tistory.com/entry/c-vector-reserve-vs-resize- ¿No puedo crear espacio con anti cipación y luego usarlo?

reserva de vectores de c++ vs cambio de ...

En el pasado, publiqué qué hacer cuando un vector aparece como a rgumento. Alguien dio su opinión en un comentario. Proporcionaré ...

codificacióndog.tistory.com

https://www.techiedelight.com/ko/find-min-or-max-value-in-a-vector-in-cpp/

Encuentre el valor mínimo o máximo del v...

Esta publicación explica cómo encontrar el valor mínimo o máximo d e un vector en C++. 1. Utilice std::max_element los iteradores std::...

www.techiedelight.com

https://cplusplus.com/reference/algorithm/max_element/?kw=max_element

max_element - Referencia de C++

plantilla de función <algoritmo> std::max_element default (1)plantilla ForwardIterator max_element (ForwardIterator primero, ForwardIter...

cplusplus.com

https://godog.tistory.com/entry/C-vector-%EC%83%9D%EC%84%B1-%EB%B0%8F-%EC%82%BD%EC%9E%85-% EC%82%AD%EC%A0%9C

Creación, inserción y eliminación de vecto...

Creación, inserción y eliminación de vectores en C++ Resultado 1 Cód igo 1 #include #include #include usando el espacio de nombres st...

	godog.tistory.com

ex02

El contenedor std::stack es bueno, pero es el único contenedor que no es iterable.

Para resolver esta injusticia, desmantelemos la pila y hagámosla iterable agregando las funcione s que queramos.

Debes crear una clase MutantStack que herede la pila y luego agregar la función iteradora.

```
int main()
{
MutantStack<int> mstack;
mstack.push(5);
mstack.push(17);
std::cout << mstack.top() << std::endl;
mstack.pop();
std::cout << mstack.size() << std::endl;
mstack.push(3);
mstack.push(5);
mstack.push(737);
//[...]
mstack.push(0);
MutantStack<int>::iterator it = mstack.begin();
MutantStack<int>::iterator ite = mstack.end();
++it;
--it;
while (it != ite)
{
std::cout << *it << std::endl;
++it;
}
std::stack<int> s(mstack);
return 0;
}
```

El comportamiento de esta prueba debería ser el mismo que std::list y es posible que deba cambi ar algunas funciones para probarla.

```cpp
template <class _Tp, class _Container /*= deque<_Tp>*/>
class _LIBCPP_TEMPLATE_VIS stack
{
public:
    typedef _Container                               container_type;
    typedef typename container_type::value_type      value_type;
    typedef typename container_type::reference       reference;
    typedef typename container_type::const_reference const_reference;
    typedef typename container_type::size_type       size_type;
    static_assert((is_same<_Tp, value_type>::value), "" );

protected:
    container_type c;
```

iterator	a random access iterator to value_type
const_iterator	a random access iterator to const value_type
reverse_iterator	reverse_iterator<iterator>
const_reverse_iterator	reverse_iterator<const_iterator>

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin	Return const_iterator to beginning (public member function)
cend	Return const_iterator to end (public member function)
crbegin	Return const_reverse_iterator to reverse beginning (public member function)
crend	Return const_reverse_iterator to reverse end (public member function)

Dado que la pila es un contenedor adaptador, si se crea una instancia sin especificar una clase de contenedor, se importa y crea una deque. Por lo tanto, puede implementar iteradores deque por tipo e implementar las funciones de cada iterador.

```cpp
template <typename T>
class MutantStack : public std::stack<T>
{
 public:
        MutantStack(void) {};
        MutantStack(const MutantStack& obj) {*this = obj;};
        MutantStack& operator=(const MutantStack& obj) {*this = obj; return (*this);}
```

```
        ~MutantStack(void) {};

        typedef typename MutantStack<T>::stack::container_type::iterator iterator;
        iterator begin(void) {return this->c.begin();}
        iterator end(void) {return this->c.end();}

        typedef typename MutantStack<T>::stack::container_type::reverse_iterator reverse_itera
tor;
        reverse_iterator rbegin(void) {return this->c.rbegin();}
        reverse_iterator rend(void) {return this->c.rend();}

        typedef typename MutantStack<T>::stack::container_type::const_iterator const_iterator;
        const_iterator cbegin(void) {return this->c.cbegin();}
        const_iterator cend(void) {return this->c.cend();}

        typedef typename MutantStack<T>::stack::container_type::const_reverse_iterator const_r
everse_iterator;
        const_iterator crbegin(void) {return this->c.crbegin();}
        const_iterator crend(void) {return this->c.crend();}
};




int main()
{
{
MutantStack<int> mstack;
mstack.push(5);
mstack.push(17);
std::cout << mstack.top() << std::endl;
mstack.pop();
std::cout << mstack.size() << std::endl;
mstack.push(3);
mstack.push(5);
mstack.push(737);
mstack.push(0);
MutantStack<int>::iterator it = mstack.begin();
MutantStack<int>::iterator ite = mstack.end();
++it;
--it;
while (it != ite)
{
std::cout << *it << std::endl;
++it;
}
std::stack<int> s(mstack);
}

std::cout << "------------------------------\n";

{
std::list<int> list;
list.push_back(5);
list.push_back(17);
std::cout << list.back() << std::endl;
list.pop_back();
std::cout << list.size() << std::endl;
list.push_back(3);
list.push_back(5);
list.push_back(737);
list.push_back(0);
std::list<int>::iterator it = list.begin();
std::list<int>::iterator ite = list.end();
```

```
++it;
--it;
while (it != ite)
{
std::cout << *it << std::endl;
++it;
}
std::list<int> s(list);
}

return 0;
}
```

Como está escrito en la tarea, la lista y la pila tienen nombres de funciones diferentes. Si crea la d eclaración principal prestando atención a esa parte, puede ver que se produce el mismo resultad o.

Otras publicaciones en la categoría ' **Programación** '

[42Seoul] ft_containers[1] - Implementación de pila (0)	2023.02.19
[42Seoul] ft_containers[0] - Resumen de la tarea (1)	2022.09.24
[42Seúl] Módulo 07 del CPP - Plantilla (1)	2022.09.11
[42Seúl] Módulo CPP 06 - Conversión de tipo (1)	2022.09.11
[42Seoul] Módulo 05 de CPP: reutilización y manejo de excepciones (3)	2022.09.10

etiqueta

#42Seúl, #cpp

Artículos relacionados con **'Programación'**

[42Seoul] ft_cont··· [42Seoul] ft_cont··· [42Seoul] Módul··· [42Seúl] Módulo ···

나의 기술부채 청산일지

어그로동크 님의 블로그입니다.

구독하기

ㅇㅇ 2024.02.24 14:40 신고 댓글주소 수정/삭제 댓글쓰기

rbegin, rend, cbegin, cend, crbegin, crend 모두 C++11 이상의 멤버함수입니다. 그대로 사용하면 subject 위반의 여지가 있지 않을까요?

nombre contraseña Secreto

Por favor ingrese sus valiosos comentarios.

Deja un comentario

Mensajes recientes

[42Seúl] ft_containers[4] - Mapa…

[42Seúl] ft_containers[3] - TR…

[42 Seúl] ft_containers[2] - Beck…

[42Seúl] ft_containers[1] - S…

[42Seúl] ft_containers[0] - y…

buscar

Por favor esc

visitantes totales

100.591

hoy	55
ayer	68