개요 | 제출 | 편집 | 코딩 결과

실습 083 - range-base for

제출 마감일: 2023-05-21 23:59
업로드 가능한 파일 수: 2
제출 방식: 개인

목적

이 실습은 range-based for 를 지원하는 클래스를 정의하는 연습을 합니다.

설명

실습 8-2 에서는 my_vector 템플릿 클래스를 정의해 보았습니다.

이번 실습에서는 우리가 정의한 my_vector 클래스가

- range-based for 문
- STL 알고리즘 함수에 사용 가능 하도록

my_vector_iterator 클래스를 정의해 봅시다.

우리는 std::vector 의 원소를 순회하기 위해서 begin(), end() 함수나

for (auto& e : vec) 와 같은 range-based for 문을 사용하였습니다.

따라서, my_vector에도 begin(), end() 멤버 함수가 필요합니다.

```
template<typename T>
class my_vector {
public:
    ...
   my_vector_iterator<T> begin(){
      return my_vector_iterator<T>(_data.get());
   }
```

```
    my_vector_iterator<T> end(){
        return my_vector_iterator<T>(_data.get()+_size);
    }
    ....
}
```

std::vector 는 begin() 멤버 함수에서

벡터의 첫번째 원소가 저장된 메모리 공간을 가리키는 iterator를 반환합니다.

이터레이터는 포인터를 추상화한 템플릿 클래스입니다.

따라서, my_vector 의 원소를 가리키는 iterator가 필요합니다.

클래스 상속을 아직 배우지 않은 관계로,

std::iterator 상속 없이 my_vector_iterator 템플릿 클래스를 정의해 보겠습니다.

```
template<typename T>
class my_vector_iterator {
public:
        my_vector_iterator(T* e=nullptr) ; // 기본 생성자
        my_vector_iterator<T>& operator++() ;  // prefix 연산자, for(auto it = vec.begin() ; it != vec.end() ; ++it )  등에 사용
         bool operator == (const my_vector_iterator<T>& e) const ;   // == 연산자 구현
         bool operator != (const my_vector_iterator<T>& e) const ;    // != 연산자 구현

private:
          T* _e = nullptr;       //my_vector 의 원소를 가리킴
}
```

문제

my_vector_iterator 템플릿 클래스를 정의하여 주어진 프로그램이 정상 동작하도록 정의하시오.

입력

없음

출력

주어진 main.cpp 파일의 표준 출력은 다음과 같다.

```
1 50 3 4 5
63
```

제출파일

83.csv

my_vector_iterator.h

참고

main.cpp ----------

```
#include "my_vector.h"
#include <numeric>
#include <iostream>

int main() {
    using namespace std;
    my_vector<int> vec = {1, 2, 3, 4, 5};
    my_vector<int> vec2(vec);
    *(++vec2.begin()) = 50;


    for (const auto& it : vec2)
        cout << it << ' ';
    cout << '\n';

    cout << accumulate(vec2.begin(), vec2.end(), 0) << '\n';
    return 0;
}
```

my_vector.h ----------

```
#ifndef TEMPLATE_MY_VECTOR_H
#define TEMPLATE_MY_VECTOR_H

#include <algorithm>
#include <cstddef>
#include <initializer_list>
#include <memory>

template<typename T>
class my_vector {
public:
    // 실습 8-2 에서 정의한 my_vector의 멤버 함수들
    // 추가 함수들 - begin(), end()
```

```
    my_vector_iterator<T> begin() {
        return my_vector_iterator<T>(_data.get());
    }

    my_vector_iterator<T> end() {
        return my_vector_iterator<T>(_data.get()+_size);
    }
private:
    size_t _size;
    std::unique_ptr<T[]> _data;
};

#endif //TEMPLATE_MY_VECTOR_H
```

my_vector_iterator.h ----------

```
#ifndef TEMPLATE_MY_VECTOR_ITERATOR_H
#define TEMPLATE_MY_VECTOR_ITERATOR_H

#include "my_vector.h"

template<typename T>
class my_vector;

template<typename T>
class my_vector_iterator {
public:
        my_vector_iterator(T* e=nullptr) ;  // 기본 생성자

 T& operator*() {
        return *_e;
    }

    my_vector_iterator<T>& operator++() {
        _e++; return *this;
    }

    bool operator == (const my_vector_iterator<T>& e) const {
        return _e == e._e;
    }

    bool operator != (const my_vector_iterator<T>& e) const {
        return !this->operator==(e);
    }


private:
         T* _e = nullptr;       //my_vector 의 원소를 가리킴
} ;


#endif //TEMPLATE_MY_VECTOR_ITERATOR_H
```