C++语言程序设计作业3

助教：叶开 ye_kai@pku.edu.cn

2022年10月26日

- 说明：完成代码的各个部分
- 如果有问题随时联系助教，包括讲义作业错误和学习上的困难
- 评分规则：
  - 每个文件都通过编译，输出无误 (5*15%=75%)
  - 代码是按照说明正确实现的 (5*5%=25%)
  - 每逾期一天，减少 10%，至多减少 50%
- 提交：
  - 截止：2022年10月25日23:59
  - 请在教学网提交代码文件，重命名为学号，例如 `2100012345.cpp` ，无需其他任何文件

Part0

- 现在，我们有了一个简单的Vector模板类，本次作业请你继续实现：
  - 在屏幕上打印Vector
  - 提供迭代器、 `begin` 和 `end` 方法
  - 提供反向迭代器、 `rbegin` 和 `rend` 方法
  - 提供一个不会轻易失效的迭代器、 `sbegin` 和 `send` 方法
- 以下是 `main.cpp` 代码，请完成Part1-Part4各个部分

```
#include <iostream>
#include <string>

template<typename T>
struct Vector {
public:
    Vector();
    Vector(int size);
```

```
    Vector(int size, const T& value);
    Vector(const Vector<T>& other);
    ~Vector();
    Vector<T>& operator=(const Vector<T>& other);
    const T& operator[](int index) const;
    T& operator[](int index);
    int size() const;
    void push_back(const T& element);
    void pop_back();
private:
    T* m_data;
    int m_capacity;
    int m_size;
};

template<typename T>
Vector<T>::Vector() {
    m_data = nullptr;
    m_capacity = 0;
    m_size = 0;
}

template<typename T>
Vector<T>::Vector(int size) {
    m_data = new T[size];
    m_capacity = size;
    m_size = size;
}

template<typename T>
Vector<T>::Vector(int size, const T& value) {
    m_data = new T[size];
    for (int i = 0; i < size; i++) {
        m_data[i] = value;
    }
    m_capacity = size;
    m_size = size;
}

template<typename T>
Vector<T>::~Vector() {
    if (m_data) delete[] m_data;
}

template<typename T>
Vector<T>::Vector(const Vector<T>& other) {
    if (other.m_size > 0) {
        m_data = new T[other.m_size];
        m_size = other.m_size;
        m_capacity = other.m_size;
        for (int i = 0; i < m_size; i++) {
            m_data[i] = other.m_data[i];
        }
    }
    else {
```

```
        m_data = nullptr;
        m_capacity = 0;
        m_size = 0;
    }
}

template<typename T>
Vector<T>& Vector<T>::operator=(const Vector<T>& other) {
    if (m_data) delete[] m_data;
    if (other.m_size > 0) {
        m_data = new T[other.m_size];
        m_size = other.m_size;
        m_capacity = other.m_size;
        for (int i = 0; i < m_size; i++) {
            m_data[i] = other.m_data[i];
        }
    }
    else {
        m_data = nullptr;
        m_capacity = 0;
        m_size = 0;
    }
    return *this;
}

template<typename T>
const T& Vector<T>::operator[](int index) const {
    return m_data[index];
}

template<typename T>
T& Vector<T>::operator[](int index) {
    return m_data[index];
}

template<typename T>
int Vector<T>::size() const {
    return m_size;
}

template<typename T>
void Vector<T>::push_back(const T& element) {
    if (m_capacity > m_size) {
        m_data[m_size] = element;
        ++m_size;
    }
    else
    {
        ++m_size;
        int new_capacity = m_capacity * 2;
        if (new_capacity < m_size) new_capacity = m_size;
        T* new_data = new T[new_capacity];
        for (int i = 0; i < m_size - 1; i++) {
            new_data[i] = m_data[i];
        }
```

```
        new_data[m_size - 1] = element;
        m_capacity = new_capacity;
        delete[] m_data;
        m_data = new_data;
    }
}

template<typename T>
void Vector<T>::pop_back() {
    if (m_size > 0) --m_size;
}

using std::cout;
using std::endl;

int main() {

    Vector<int> a;
    Vector<std::string> b;

    for (int i = 0; i < 5; ++i) {
        a.push_back(i + 1);
    }

    for (int i = 0; i < 6; ++i) {
        b.push_back(std::to_string(i + 1));
    }


    // now a = { 1, 2, 3, 4, 5 }
    // now b = { "1", "2", "3", "4", "5", "6" }

    cout << "----------------" << endl;

    // 1. cout, ostream

    cout << "1. cout, ostream" << endl;

    cout << a << endl; // [ 1 2 3 4 5 ]
    cout << b << endl; // [ 1 2 3 4 5 6 ]

    cout << "----------------" << endl;




    cout << "2. begin, end" << endl;

    cout << "[ ";
    for (auto itor = a.begin(); itor != a.end(); ++itor) {
        cout << *itor << " "; // [ 1 2 3 4 5 ]
    }
    cout << "]" << endl;

    cout << "[ ";
```

```
    for (auto i : b) {
        cout << i << " "; // [ 1 2 3 4 5 6 ]
    }
    cout << "]" << endl;

    cout << "----------------" << endl;




    cout << "3. rbegin, rend" << endl;

    cout << "[ ";
    for (auto itor = a.rbegin(); itor != a.rend(); ++itor) {
        cout << *itor << " "; // [ 5 4 3 2 1 ]
    }
    cout << "]" << endl;

    cout << "----------------" << endl;




    cout << "4 sbegin, send" << endl;

    cout << "[ ";
    for (auto itor = a.sbegin(); itor != a.send(); ++itor) {
        cout << *itor << " "; // [ 1 3 5 ]
    }
    cout << "]" << endl;

    cout << "[ ";
    for (auto itor = b.sbegin(); itor != b.send(); ++itor) {
        cout << *itor << " "; // [ 1 3 5 ]
    }
    cout << "]" << endl;

    cout << "----------------" << endl;




    cout << "5. safe sbegin, send" << endl;

    Vector<int> dummy1 = a;
    cout << "[ ";
    for (auto itor = a.ssbegin(); itor != a.ssend(); ++itor) {
        a = dummy1;
        cout << *itor << " "; // [ 1 3 5 ]
    }
    cout << "]" << endl;

    Vector<std::string> dummy2 = b;
    cout << "[ ";
    for (auto itor = b.ssbegin(); itor != b.ssend(); ++itor) {
```

```
        for (int i = 0; i < 5; ++i) b.push_back(std::to_string(i));
        for (int i = 0; i < 5; ++i) b.pop_back();
        cout << *itor << " "; // [ 1 3 5 ]
    }
    cout << "]" << endl;

    cout << "----------------" << endl;

    return 0;
}
```

Part1

- 重载 `std::ostream` 的流输出运算符，使得 `std::cout` 也可以通过 `<<` 打印数组的内容
- 注意：你的运算符不仅要对 `std::cout` 适用，也要对任何 `std::ostream` 适用（ `std::cout` 是一种 `std::ostream` ），这样才是一个完整的实现
  - 可以参照之前的代码
- 输出：

```
----------------
1. cout, ostream
[ 1 2 3 4 5 ]
[ 1 2 3 4 5 6 ]
----------------
```

Part2

- 实现Vector类的迭代器方法 `begin` 和 `end` ，使它就像 `std::vector` 那样工作
- 我们知道，指针本身就是一种迭代器，因此你可以直接返回指针，即 `T*` 类型
- 输出：

```
----------------
2. begin, end
[ 1 2 3 4 5 ]
[ 1 2 3 4 5 6 ]
----------------
```

Part3

- 实现Vector类的迭代器方法 `rbegin` 和 `rend` ，使它就像 `std::vector` 那样工作
- 这两个方法将返回一组倒序遍历的迭代器，这个时候我们就不能直接返回指针了（因为指针的 `++` 是正序的），必须自己实现一个结构体，例如叫做 `ReverseIterator` ，才能达到 `++` 操作反而反向移动的效果
- 我们应该定义一个结构体：

```
struct ReverseIterator {
    // ...
};
```

思考：这个定义应该在Vector类内还是在类外？

- 注意，`Vector<T>` 是模板类，那么它的迭代器自然也是随着 `T` 的类型的变化而变化的
- 所以这个定义必须在Vector内
- 当然如果定义到Vector外，也是可以的，但是它就必须适配所有类型 `T` ，因此要这样声明：

```
template<typename T>
struct ReverseIterator {
    // ...
};
```

- 然后，思考，`ReverseIterator` 需要什么成员变量？
  - 我们可以在内部记录一个指针 `T*` ，然后每次需要 `++` 的时候，就让这个内部指针 `--`
- 最后，`ReverseIterator` 需要什么成员方法？
  - 我们不需要实现完整的运算符，只需要 `main` 函数中用到的那些

```
template<typename T>
struct Vector {
    struct ReverseIterator {
        // ...

        void operator++();
        T* operator*();
        bool operator!=(const ReverseIterator & other);
    };

    // ...

    ReverseIterator begin();
    ReverseIterator end();

    // ...
};
```

- 输出：

```
---------------
3. rbegin, rend
[ 5 4 3 2 1 ]
---------------
```

Part4

- 实现Vector类的迭代器方法 `sbegin` 和 `send` ，我们自定义一种迭代方式，是“跳跃”地迭代
  - 具体来说，每次递增都将额外跳过一个元素

- 同样地，我们必须自己实现一种迭代器结构体，例如 `SkipIterator`，然后自定义它的 `++` 功能
- 特别注意：`send` 应该返回什么？
  - 我们的 `for` 循环终止条件是 `itor == a.send()`，注意，不是 `itor >= a.send()`
  - 因此，如果 `a.send()` 是奇数位置，而 `itor` 是偶数位置，那么它们永远不会相等
  - 请谨慎地设计 `send`
- 输出：

```
----------------
4 sbegin, send
[ 1 3 5 ]
[ 1 3 5 ]
----------------
```

Part5

- 你是否了解迭代器失效？
  - 将Part5中 `ssbegin` 和 `ssend` 改成 `sbegin` 和 `send`，试一试运行这个代码
    - 是否出现错误？（如果没出现错误而且你不明白迭代器失效，请联系助教）
  - 思考：为什么发生这样的错误
    - 我们在 `SkipIterator` 中存的是指针，此时如果 `Vector` 中，因为数据增删改变数组大小，而导致 `m_data` 重新分配了，那么 `SkipIterator` 中存的指针会指向什么地方？
- 因此，我们在Part5中要实现一个 `SafeSkipIterator` 以及对应的 `ssbegin`、`ssend`，能够避免迭代器失效带来的问题
  - 思考：如何实现？
    - 我们不能存 `m_data` 相关的数据，因为它可能随时重新分配
  - 那么，能否存 `Vector` 本身的指针呢？然后通过记录偏移量去访问数据
    - 因为，`a[10]` 是不会失效的，但是 `a.m_data[10]` 是会失效的
- 输出：

```
----------------
5. safe sbegin, send
[ 1 3 5 ]
[ 1 3 5 ]
----------------
```