vector

vector的定义

- **vector()**
  - 无参构造函数，构造一个空的vector
- vector（size_type num, const value_type& val = value_type())
  - 构造一个初始放入num个值为val的元素的Vector
- **vector(const vector& from)**
  - 拷贝构造函数

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    //构造函数
    vector<int> v1;
    //构造5个容器并初始化为0
    vector<int> v2(5, 0);
    //拷贝构造函数
    vector<int> v3(v2);
    return 0;
}
```

vector的遍历

- operator[ ]
  - 返回pos位置的字符
- **iterator begin()+ iterator end()**
  - begin返回一个指向当前vector起始元素的迭代器
  - end返回一个指向当前vector末尾元素的**下一位置**的迭代器；如果你要访问末尾元素，需要先将此迭代器自减1
- **reverse_rbegin() + reverse_rend()**
  - rbegin返回指向当前vector末尾的逆迭代器(实际指向末尾的下一位置，而其内容为末尾元素的值)
  - rend返回指向当前vector起始位置的逆迭代器
- 范围for
  - C++11支持更简洁的范围for的新遍历方式

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
```

```
    //构造函数
    vector<int> v1;

    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v1.push_back(4);

    //1.operator[]+size()
    for (size_t i = 0; i < v1.size(); ++i)
    {
        cout << v1[i] << " ";
    }
    cout << endl;

    //2.迭代器
    //begin+end
    vector<int>::iterator it = v1.begin();
    while (it != v1.end())
    {
        cout << *it << " ";
        ++it;
    }
    cout << endl;

    //rbegin+rend
    vector<int>::reverse_iterator rit = v1.rbegin();
    while (rit != v1.rend())
    {
        cout << *rit << " ";
        ++rit;
    }
    cout << endl;

    //3.范围for——本质是由编译器替换成迭代器方式遍历支持的
    for (auto e : v1)
    {
        cout << e << " ";
    }
    cout << endl;

    return 0;
}
```

vector的容量

- size_type size() const
  - 返回当前vector所容纳元素的数目
- size_type capacity() const
  - 返回当前vector在重新进行内存分配以前所能容纳的元素数量
- bool empty()
  - 如果当前vector没有容纳任何元素，则empty()函数返回true，否则返回false
- **void resize(size_type size, value_type val = value_type())**
  - 改变当前vector的大小为size,且对新创建的元素赋值val

    - 如果size小于当前容器的大小，则将内容减少到其前size个元素，并删除超出范围的元素（并销毁它们）
    - 如果size大于当前容器的大小，则通过在末尾插入所需数量的元素来扩展内容，以达到size。如果指定了*val*，则将新元素初始化为val
    - 如果size也大于当前容器容量，将自动重新分配已分配的存储空间
- **reserve**
    - 为当前vector预留至少共容纳size个元素的空间

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    //构造函数
    vector<int> v1;

    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v1.push_back(4);

    //size()
    cout << v1.size() << endl;

    //capacity()
    cout << v1.capacity() << endl;

    //empty()
    cout << v1.empty() << endl;

    //resize()
    v1.resize(8, 0);
    cout << v1.size() << endl;

    //reserve()
    v1.reserve(10);
    cout << v1.capacity() << endl;

    return 0;
}
```

vector的增删查改

- **void push_back(const value_type& val)**
    - 添加值为val的元素到当前vector末尾
- **void pop_back()**
    - 删除当前vector最末的一个元素
- iterator insert(iterator position, const value_type& val)
    - 在指定位置position前插入值为val的元素,返回指向这个元素的迭代器
- iterator erase(iterator position)
    - 删除指定位置position的元素，返回删除元素的下一位置的迭代器
- void swap(vector& x)

    - 交换当前vector与vector x的元素
- find
    - 查找(这个是算法模块实现，不是vector的成员接口),返回一个迭代器
- sort
    - 升序排序，底层用快排实现

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
    //构造函数
    vector<int> v1;

    //push_back(val)
    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v1.push_back(4);

    //pop_back()
    v1.pop_back();

    //insert(pos,val)
    v1.insert(v1.begin(), 0);
    for (auto e : v1)
    {
        cout << e << " ";
    }
    cout << endl;

    //erase(pos)
    v1.erase(v1.begin());
    for (auto e : v1)
    {
        cout << e << " ";
    }
    cout << endl;

    //删掉4
    //find
    vector<int>::iterator pos = find(v1.begin(), v1.end(), 2);
    if (pos != v1.end())
    {
        v1.erase(pos);
    }
    for (auto e : v1)
    {
        cout << e << " ";
    }
    cout << endl;

    //sort
    vector<int> v2;
```

```
    v2.push_back(10);
    v2.push_back(40);
    v2.push_back(30);
    v2.push_back(20);
    sort(v2.begin(), v2.end());
    for (auto e : v2)
    {
        cout << e << " ";
    }
    cout << endl;

    //swap
    vector<int> v3;
    v3.push_back(1);
    v3.push_back(2);
    v3.push_back(3);
    v3.push_back(4);
    v3.swap(v2);
    for (auto e : v3)
    {
        cout << e << " ";
    }
    cout << endl;

    return 0;
}
```

vector的模拟实现

```
#pragma once
#include<cstring>
#include<cassert>
#include<iostream>
using namespace std;

namespace bin {
    template<class T>
    class vector
    {
    public:
        typedef T* iterator;
        typedef const T* const_iterator;

        //构造函数
        vector()
            :_start(nullptr)
            , _finish(nullptr)
            ,end_of_storage(nullptr)
        {}

        //拷贝构造
        //vector(const vector<T>& v)
        //{
        //  //深拷贝
        //  //开空间
        //  _start = new T[v.size()];
```

```
//  _finish = _start + v.size();
//  end_of_storage = _start + v.capacity();
//  //拷贝数据
//  memcpy(_start, v._start, sizeof(T) * v.size());
//}
vector(const vector<T>& v)
    :_start(nullptr)
    ,_finish(nullptr)
    ,end_of_storage(nullptr)
{
    reserve(v.capacity());
    for (auto& e : v)
    {
        push_back(e);
    }
}

//赋值运算符重载
vector<T>& operator=(vector<T> v)
{
    this->swap(v);
    return *this;
}

//swap()
void swap(vector<T>& v)
{
    ::swap(_start, v._start);
    ::swap(_finish, v._finish);
    ::swap(end_of_storage, v.end_of_storage);
}

//析构函数
~vector()
{
    delete[] _start;
    _start = _finish = end_of_storage = nullptr;
}

//operator[]运算符重载
T& operator[](size_t i)
{
    assert(i < size());
    return _start[i];
}

//begin()返回一个指向当前vector起始元素的迭代器
iterator begin()
{
    return _start;
}
const_iterator begin() const
{
    return _start;
}

//end()返回一个指向当前vector末尾元素的下一位置的迭代器
iterator end()
```

```
        {
            return _finish;
        }
        const_iterator end() const
        {
            return _finish;
        }

        //reserve()为当前vector预留至少共容纳n个元素的空间
        void reserve(size_t n)
        {
            if (n > capacity())
            {
                size_t sz = size();
                //开辟新空间
                T* tmp = new T[n];
                if (_start)
                {
                    //拷贝数据
                    memcpy(tmp, _start, sizeof(T) * sz);
                    //释放旧空间
                    delete[] _start;
                }
                //指向新空间
                _start = tmp;
                //重新计算_finish和end_of_storage
                _finish = tmp + sz;
                end_of_storage = _start + n;
            }
        }

        //resize()改变当前vector的大小为size,且对新创建的元素赋值val
        void resize(size_t n, const T& val = T())
        {
            //n小于当前容器的大小size()，则将内容减少到其前n个元素
            if (n < size())
            {
                _finish = _start + n;
            }
            else
            {
                if (n > capacity())
                {
                    //如果n大于当前容器容量，将重新分配存储空间
                    reserve(n);
                }
                //n大于当前容器的大小,在末尾插入元素来达到n
                //如果指定了val，则将新元素初始化为val
                while (_finish != _start + n)
                {
                    *_finish = val;
                    ++_finish;
                }
            }
        }

        //push_back()添加值为x的元素到当前vector末尾
        void push_back(const T& x)
```

```
    {
        /*if (_finish == end_of_storage)
        {
            size_t newcapacity = capacity() == 0 ? 2 : capacity() * 2;
            reserve(newcapacity);
        }
        *_finish = x;
        ++_finish;*/

        insert(end(), x);
    }

    //pop_back()删除当前vector最末的一个元素
    void pop_back()
    {
        /*assert(_start < _finish);
        --_finish;*/

        erase(end() - 1);
    }

    //insert()插入
    void insert(iterator pos, const T& x)
    {
        assert(pos <= _finish);
        if (_finish == end_of_storage)
        {
            //内部迭代器失效，则先计算出来pos到_start的距离
            size_t n = pos - _start;
            size_t newcapacity = capacity() == 0 ? 2 : capacity() * 2;
            reserve(newcapacity);
            //增容后pos迭代器失效，在新申请的空间中重新计算pos的位置
            pos = _start + n;
        }
        //pos之后的元素往后移
        iterator end = _finish - 1;
        while (end >= pos)
        {
            *(end + 1) = *end;
            --end;
        }
        //将x插入pos位置
        *pos = x;
        //当前vector的大小加1
        ++_finish;
    }

    //erase()删除
    iterator erase(iterator pos)
    {
        assert(pos < _finish);
        iterator it = pos;
        while (it < _finish)
        {
            *it = *(it + 1);
            ++it;
        }
        --_finish;
```

```
            //返回删除之后的下一个位置
            return pos;
        }

        //size()返回当前vector所容纳元素的数目
        size_t size() const
        {
            return _finish - _start;
        }

        //capacity()返回当前vector在重新进行内存分配以前所能容纳的元素数量
        size_t capacity() const
        {
            return end_of_storage - _start;
        }

    private:
        iterator _start;
        iterator _finish;
        iterator end_of_storage;
    };
}
```