

Cài đặt đơn giản lớp Vector

Sử dụng lớp `vector` của thư viện `std`

Một trong những kiểu hay được sử dụng nhất trong thư viện `std` là `vector`. Một `vector` là một dãy các phần tử có cùng kiểu. Các phần tử được lưu trữ liên tục trong bộ nhớ.

Ví dụ (chạy với C++11): Hàm `push_back()` để thêm một số phần tử vào cuối danh sách.

```
#include <vector>
#include <iostream>
using namespace std;
int main(int argc, char const *argv[]) {
    vector<int> v;
    v.push_back(1);    v.push_back(2);
    v.push_back(1);    v.push_back(2);
    v.push_back(3);    v.push_back(4);

    for (int i = 0; i < v.size(); i++)
        cout<<v[i]<<" ";    // Thao tác giống như mảng
    cout<<endl;

    for (auto x: v) {        // Cấu trúc for theo dãy
        cout<<x<<" ";
    }
    return 0;
}
```

Cấu trúc `for` theo dãy cho phép in các phần tử trong `vector` ra màn hình theo cách đơn giản (không cần chỉ số như mảng).

Để biên dịch file nguồn `testvector.cpp` với **C++11** ta dùng lệnh.

```
→ Sources g++ -std=c++11 testvector.cpp -o vidu
→ Sources ./vidu
1 2 1 2 3 4
1 2 1 2 3 4
```

Ví dụ: Khởi tạo `vector`

```
vector<int> v1 = {1, 2, 3, 4};    // kích thước là 4
vector<string> v2;                // kích thước là 0
vector<int*> v3(23);              // kích thước là 23; giá trị khởi tạo: nullptr
vector<double> v4(32, 9.9);      // kích thước là 32; giá trị khởi tạo: 9.9
```

Mặc định các giá trị trong `vector` sẽ được khởi tạo là `0` cho giá trị số và là con trỏ `nullptr` cho con trỏ. Nếu bạn không muốn giá trị mặc định, bạn có thể xác định giá trị trong tham số thứ hai.

Ví dụ: Xóa phần tử cuối danh sách dùng hàm `pop_back()` và xóa phần tử ở vị trí bất kỳ dùng `erase()`.

```
v.pop_back();
v.erase(v.begin());           // Xóa phần tử đầu tiên
v.erase(v.begin() + v.size() - 1); // Xóa phần tử cuối cùng
```

Bài tập: Hãy tìm hiểu các hàm sau trong lớp `vector`.

Hàm	Ý nghĩa và cách sử dụng
<code>v.size();</code>	
<code>v.front();</code>	
<code>v.back()</code>	
<code>v.empty()</code>	
<code>v.clear();</code>	
<code>v.resize()</code>	

Ví dụ: Sắp xếp các phần tử trong `vector` theo thứ tự tăng dần

```
#include <algorithm>
sort(v.begin(), v.end());
```

hoặc sắp xếp giảm dần với hàm `greater`. Hàm `greater` so sánh hai đối tượng, hàm này trả về giá trị `true` nếu đối tượng thứ nhất lớn hơn đối tượng thứ hai.

```
#include <algorithm>
sort(v.begin(), v.end(), greater<int>());
```

Cài đặt đơn giản

Ta thử cài đặt lớp `vector` một cách đơn giản như sau:

```
class Vector {
public:
    Vector(int s) {
        elem = new double[s];
        sz = s;
    }
    ~Vector() { delete [] elem; }
    double& operator[](int i) { return elem[i]; }
    int size() { return sz; }
private:
    double* elem; // con trỏ tới phần tử đầu của mảng
    int sz;       // số phần tử
};
```

Hàm sau đây để tính tổng dãy số với lớp `vector` cài đặt ở trên.

```
double read_and_sum(int s) {
    vector v(s); // tạo một vector gồm s phần tử
    for (int i=0; i!=v.size(); ++i)
        cin>>v[i]; // đọc các phần tử
    double sum = 0;
    for (int i=0; i!=v.size(); ++i)
        sum+=v[i]; // tính tổng các phần tử
    return sum;
}
```

Cấu tử khởi tạo với danh sách

Kiểu dữ liệu `std::initializer_list` được dùng để định nghĩa **cấu tử khởi tạo với danh sách**: khi ta dùng danh sách với `{}`, ví dụ như danh sách `{1,2,3,4}`, trình biên dịch sẽ tạo ra một đối tượng với kiểu `initializer_list` để đưa vào chương trình. Vậy, ta có thể viết

```
vector v1 = {1,2,3,4,5}; // v1 có 5 phần tử
vector v2 = {1.23, 3.45, 6.7, 8}; // v2 có 4 phần tử
```

Cấu tử khởi tạo với danh sách có thể được định nghĩa như sau: sau:

```
vector::vector(initializer_list<double> lst) // khởi tạo vector với một danh sách
{
    elem = new double[lst.size()];
    sz = lst.size();
    copy(lst.begin(), lst.end(), elem); // copy từ lst vào elem
}
```

Bài tập: Thêm các hàm sau vào lớp `vector`

- Hàm `void push_back (const double& n)` thêm một phần tử vào cuối dãy. Với cài đặt này, ta có thể viết, ví dụ:

```
vector read() {
    vector v;
    for (double d; cin>>d;)
        v.push_back(d);
    return v;
}
```

- Hàm `void pop_back ()` xóa phần tử cuối cùng của dãy;
- Hàm `void erase (int i)` xóa phần tử thứ `i` trong danh sách;
- Hàm `void resize(int n)` để thay đổi kích thước của vector. Sau khi `resize`, giá trị các phần tử không bị thay đổi.

Kiểu được tham số hóa với `template`

Chúng ta có thể tổng quát hóa các `vector` kiểu `double` thành `vector` có kiểu bất kỳ dùng `template` và thay thế kiểu `double` với tham số. Ví dụ:

```
template<typename T>
class Vector {
private:
    T* elem;          // elem trỏ tới mảng gồm sz phần tử có kiểu T
    int sz;
public:
    Vector(int s);
    ~Vector() { delete[] elem; }
    // ...
    T& operator[](int i);
    int size() const { return sz; }
};
```

Vòng lặp `for` theo dãy

Để lớp `vector` có thể lặp theo dãy theo cấu trúc `for (auto & x: Set)` ta phải thêm hai hàm `begin()` và `end()` để chỉ ra **địa chỉ của phần tử bắt đầu** và **địa chỉ phần tử kết thúc** trong dãy. Cài đặt hai hàm này như sau.

```
template<typename T>
T * Vector::begin() {
    return &elem[0];
}
template<typename T>
T * Vector::end() {
    return begin() + sz;
}
```

Từ đó ta có thể sử dụng cấu trúc `for` theo dãy và hàm sắp xếp `sort`.

```
int main(int argc, char const *argv[]) {
    Vector<double> s(4);
    s[0] = 2;   s[1] = 1;
    s[2] = 4;   s[3] = 0;
    sort(s.begin(), s.end());
    for (auto& i : s)
        cout<<i<<endl;
    return 0;
}
```

Mã nguồn đầy đủ cho lớp `Vector` đơn giản

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

template<typename T>
class Vector
{
private:
    T *elem;
    int sz;
public:
    Vector (int s) {
        elem = new T[s];
        sz = s;
    }

    // khởi tạo Vector với một danh sách
    Vector(initializer_list <T> lst) {
        elem = new T[lst.size()];
        sz = lst.size();
        copy(lst.begin(),lst.end(),elem);    // copy từ lst vào elem
    }

    T& operator[] (int i){return elem[i];}
    T * begin() {return &elem[0];}
    T * end() {return begin() + sz;}
};

int main(int argc, char const *argv[])
{
    Vector <double> s = {2, 1, 3, 0};
    sort(s.begin(),s.end());
    for (auto& i : s)
        cout<<i<<" ";
    return 0;
}
```

Biên dịch và chạy với **C++11** như sau.

```
→ Sources g++ -std=c++11 SVector.cpp -o SVector
→ Sources ./SVector
0 1 2 3
```