Entrada e Saída

Contêineres

Iteradores

Algoritmos

Referências

C++ Survival Kit

(for Programming Challenges)

Daniel K. O.

March 19, 2010

Roteiro

Entrada e Saída

Contêineres

Iteradores

Algoritmos

Referências

Leitura simples

```
#include <iostream> // cin, cout, endl


int a; float b;
cin >> a >> b;
if (cin)
  cout << a << " " << b << endl;

if (cin >> a >> b)
    cout << "leitura funcionou" << endl;

if ( ! (cin >> a >> b) )
    cout << "leitura falhou" << endl;
```

Lendo linhas inteiras

```
#include <string> // string, getline()


string linha;
getline(cin, linha); // le uma linha inteira

while (getline(cin, linha)) // linha por linha
    // ... processa a linha
```

```
cin >> a;
getline(cin, linha);
```

Problema:

``5 \n asdf''

Lendo linhas inteiras

```
#include <string> // string, getline()


string linha;
getline(cin, linha); // le uma linha inteira

while (getline(cin, linha)) // linha por linha
    // ... processa a linha

cin >> a;
getline(cin, linha);
```

Problema:

```
``5 \n asdf''
```

Leitura mista - versão 1

```
#include <stdio.h> // sscanf()

int i; char c; float f; double d; unsigned u;
string linha;
getline(cin, linha);
if ( sscanf(linha.c_str() , "%d  %c  %f %lf  %u",
                            &i, &c, &f, &d, &u
           ) == 5 ) {
    // ok
} else {
    // falhou
}
```

Obs: A string de formato de leitura é bem poderosa.

Leitura mista - versão 2

```
#include <sstream> // para istringstream

int i; char c; float f; double d; unsigned u;
string linha;
getline(cin, linha);
istringstream is(linha);
if ( is >> i >> c >> f >> d >> u ) {
    // ok
} else {
    // falhou
}

while (is >> i) {
    // processa i
}
```

Leitura mista - versão 2

```
#include <sstream> // para istringstream

int i; char c; float f; double d; unsigned u;
string linha;
getline(cin, linha);
istringstream is(linha);
if ( is >> i >> c >> f >> d >> u ) {
    // ok
} else {
    // falhou
}

while (is >> i) {
    // processa i
}
```

Leitura mista - versão 3

```
/* Entrada:
     a\n
     linha\n
 */
int a; b;
char dummy;
string linha;

cin >> a;
cin >> dummy;
cin.putback(dummy);
getline(cin, linha);
```

Impressão com separador

```
/* v[0], v[1], v[2], ..., v[n-1] */
int v[N];

const char *sep = "";
for (int i=0; i<N; ++i, sep=", ")
    cout << sep << v[i];
```

Índices de arrays arbitrários

```
int x[20] = { ... };
int *y = x+10;

for (int i=-10; i<10; ++i)
    cout << y[i] << endl;
```

STL

A Standard Template Library é uma biblioteca de algoritmos de estruturas de dados baseada no paradigma de programação genérica.
É composta de 3 elementos-chave:

- Contêineres
- Algoritmos
- Iteradores

Contêineres

```
vector<int> a;
list<string> b;
deque<double> c;

map<string, int> d;
multimap<string, int> e;
set<string> f;
multiset<string> g;

string h;
bitset<128> i;
```


Entrada e Saída
Contêineres
Iteradores
Algoritmos
Referências

vector

```
vector<int> a(5, -1); // { -1, -1, -1, -1, -1 }

int v[4] = {1, 2, 3, 4};
vector<int> b(v, v+4);
b.push_back(7);
cout << b.size() << endl;
cout << b[2] << endl; // imprime 3
```

map

```
map<string, int> dict;

dict["um"] = 1;
dict["dois"] = 2;

cout << dict["um"] << endl;
cout << dict["tres"] << endl;
```

set, multiset

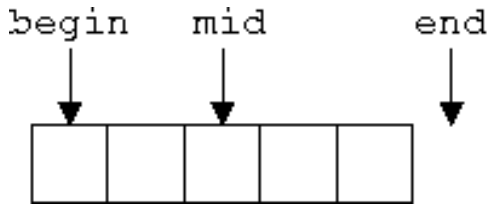
```
set<string> a;
multiset<string> b;

a.insert("azul");
a.insert("azul"); // ignorado
a.insert("verde");

b.insert("azul");
b.insert("azul");
b.insert("verde");

cout << a.size() << endl; // 2
cout << b.size() << endl; // 3

cout << a.count("azul") << endl; // 1
cout << b.count("azul") << endl; // 2
```

Iteradores

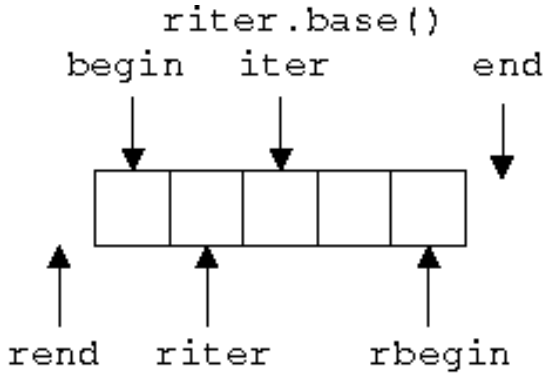
Um iterador é qualquer coisa que represente uma posição em um contêiner. Deve permitir percorrimento (`++i, --i, i+=3, i-=2`) e acesso (`*i`).

- .begin(), .end()
- .rbegin(), .rend()
- **istream_iterator**, **ostream_iterator**
- ponteiros para arrays nativos.

Intervalos



- [begin, mid) [mid, end) = [begin, end)
- [i, i) = vazio



list

```
list<string> words;
string s;

while (cin >> s)
    words.push_back(s);

list<string>::iterator i;
for (i=words.begin(); i!=words.end(); ++i)
    cout << *i << " : " << i->length() << endl;
```

Algoritmos

Entrada e Saída
Contêineres
Iteradores
Algoritmos
Referências

Cabeçalhos:

- <algorithm>: `count`, `find`, `search`, `for_each`, `lower_bound`, `max`, `max_element`, `copy`, `fill`, `merge`, `sort`, `partial_sort`, `nth_element`, `partition`, `stable_sort`, `unique`, `rotate`, `reverse`, `swap_ranges`, `next_permutation`, `make_heap`, `pop_heap`, etc.
- <numeric>: `accumulate`, `inner_product`, `partial_sum`, `adjacent_difference`
- <functional>: `less`, `greater`, `plus`, `multiplies`, etc.

Exemplo: vector, iteradores

```
istream_iterator<string> a(cin), b;
vector<string> v(a, b);

sort(v.begin(), v.end());

vector<string>::iterator novo_fim =
    unique(v.begin(), v.end());
v.erase(novo_fim, v.end());

copy(v.begin(), v.end(),
    ostream_iterator<string>(cout, " : "));
cout << endl;

vector<string>::iterator x =
    lower_bound(v.begin(), v.end(), "p");
if (x != v.end())
    cout << *x << endl;
```

Exemplo: separar pares de ímpares

```
bool isEven(int x)
{
    return x % 2 == 0;
}

vector<int> v = ...;

vector<int>::iterator impares =
    partition(v.begin(), v.end(), isEven);

copy(v.begin(), impares,
    ostream_iterator<int>(cout, " : "));
cout << endl;
copy(impares, v.end(),
    ostream_iterator<int>(cout, " : "));
cout << endl;
```

Exemplo: ordem decrescente

```
#include <functional> // greater<>

bool maior(int a, int b)
{
    return a > b;
}

vector<int> v = ...;

sort(v.begin(), v.end(), maior);

sort(v.begin(), v.end(), greater<int>());

sort(v.rbegin(), v.rend());
```

Exemplo: contador de freqüências

```
string w;
map<string,int> contador;

while (cin >> w)
    contador[w]++;

map<string,int>::iterator i;

for (i=contador.begin(); i!=contador.end(); ++i)
    cout << i->first << " : "
         << i->second << endl;
```

Exemplo: comparando sequências

```
vector<int> a = ..., b = ...;

if (a.size() == b.size() and
    equal(a.begin(), a.end(), b.begin())

      cout << "sequencias iguais" << endl;

else {
    if (a.size() > b.size())
        swap(a, b);

    vector<int>::iterator i =
        mismatch(a.begin(), a.end(), b.begin());

    cout << "primeira diferenca na posicao " <<
        distance(a.begin(), i) << endl;
}
```

Referências

- Web:
  - Rogue Wave C++ Library Reference
  - GNU libstdc++
- Livro: The C++ Programming Language