Standard Template Library

Associativos
Iterators
Algoritmos

Laboratório de Programação Competitiva I

Rene Pegoraro

Pedro Henrique Paiola

Wilson M Yonezawa

set

- Armazenam elementos únicos seguindo uma ordem.
  - um critério de comparação precisa ser definido

```
#include <string>
#include <set>
using namespace std;

int main() {
  set <string> nome;
  nome.insert("joao");
  nome.insert("maria");
  nome.insert("jose");
  nome.insert("pedro");
  nome.insert("maria");
  nome.insert("joao");
  printf("%d\n", nome.size());
  return 0;
}
```

- O set não armazena elementos repetidos
- No exemplo ao lado, o valor apresentado será 4 (número de elementos no set)
- Se precisar de armazenar repetições, multiset poderá ser usado
- Veja também hash_set

set - Formas de Ordenação

```
#include <string>
#include <set>
using namespace std;

struct Par {
  int a, b;
  Par(int A, int B) : a(A), b(B) {}
  bool operator<(const Par &p) const {
    return b < p.b;
  }
};
struct comp {
  bool operator()(const Par &p1,
                  const Par &p2) {
    return p1.a < p2.a;
  }
};
```

```
int main() {
  set<Par> conj1;
  conj1.insert(Par(1, 91));
  conj1.insert(Par(2, 52));
  conj1.insert(Par(3, 73));
  conj1.insert(Par(1, 71));
  set<Par, comp> conj2;
  conj2.insert(Par(1, 91));
  conj2.insert(Par(2, 52));
  conj2.insert(Par(3, 73));
  conj2.insert(Par(1, 71));
  printf("%ld, %ld\n",
         conj1.size(), conj2.size());
  return 0;
}
```

Ordem alternativa indicada pelo “operator()”.

Exercício Usando set

- Resolva o problema URI 2174
- Recomendações
  - Use set<string> para definir o conjunto de pomekons.
  - Como o conjunto não aceita repetições, basta inserir todos os pokemons na lista, mesmo se repetidos, pois eles serão inseridos apenas uma vez.

map

- Mantém pares chave-valor, sendo cada chave única
  - Precisa de um critério de comparação entre as chaves.

```
#include <cstdio>
#include <map>
#include <string>
using namespace std;

int main () {
  map<string,float> nota;
  nota["Pedro"]=10.0;
  nota["Antonio"]=5.0;
  nota["Maria"]=7.5;
  printf("%f\n", nota["Pedro"]);
  printf("%f\n", nota["pedro"]);
  printf("%f\n", nota["Maria"]);
  return 0;
}
```

- O dicionário não armazena chaves repetidas
- O container multimap armazena repetições
- Usar hash_map e hash_multimap pode melhorar o desempenho mas sem ordenação

Exercício Usando map

- Resolva o problema URI 2727
- Recomendações
  - Use map<string,char> para definir a relação chave-valor, assim a chave será o código e o valor o caractere
  - A iniciação do map pode ser feito manualmente, adicionando os 26 códigos um a um para cada caractere ou programaticamente.
    - Definindo:
      - **map<string, char> decodifica;**
    - Manualmente para a letra “i”, seria:
      - **decodifica["... ... ..."] = 'i';**
  - Para decodificar, basta ler o código de cada linha em uma string e usá-la como chave

Iteradores (iterator)

- Um iterator é qualquer objeto capaz de interagir com os elementos de um container usando um conjunto de operadores
- Uma variável iterator funciona como um ponteiro, “apontando” para cada elemento do container
- Um iterator pode ser incrementado, para indicar o próximo elemento

Iterator Mimetiza Aritmética de Ponteiros

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

float vet[] = {7.8, 1.2, 5.6, 9.0, 3.4};
vector<float> vect(&vet[0], &vet[5]);

int main() {
  for (vector<float>::iterator it = vect.begin();
       it != vect.end(); it++) {
    printf("%5.1f", *it);
  }
  printf("\n");
  for (float *it = &vet[0]; it != &vet[5]; it++) {
    printf("%5.1f", *it);
  }
  printf("\n");
```

No vídeo, obtém-se:

```
 7.8  1.2  5.6  9.0  3.4
 7.8  1.2  5.6  9.0  3.4
 7.8  1.2  5.6  9.0  3.4
 7.8  1.2  5.6  9.0  3.4
```

```
  for (float f: vect) {
    printf("%5.1f", f);
  }
  printf("\n");
  for (float f: vet) {
    printf("%5.1f", f);
  }
  printf("\n");
  return 0;
}
```

range-based for
a partir do C++11

Iterator com set

```
#include <string>
#include <set>
using namespace std;
int main() {
  set<string> nome;
  nome.insert("joao");
  nome.insert("maria");
  nome.insert("jose");
  nome.insert("pedro");
  nome.insert("maria");
  nome.insert("joao");
  set<string>::iterator iter;
  for (iter = nome.begin(); iter != nome.end(); iter++) {
    printf("%s\n", iter->c_str());
  }
  return 0;
}
```

No vídeo, obtém-se:
joao
jose
maria
pedro

Iterator com set

```
#include <string>
#include <set>
using namespace std;

struct Par {
  int a, b;
  Par(int A, int B) {
    a = A;
    b = B;
  }
  bool operator<(const Par &p)
  const {
    return a < p.a;
  }
};
```

```
int main() {
  set<Par> conj;
  conj.insert(Par(1, 91));
  conj.insert(Par(2, 52));
  conj.insert(Par(3, 73));
  conj.insert(Par(1, 71));
  for (set<Par>::iterator it =
       conj.begin();
       it != conj.end(); it++) {
    printf("%d\t%d\n", it->a, it->b);
  }
  return 0;
}
```

No vídeo, obtém-se:

```
1    91
2    52
3    73
```

Iterator com multiset

```
#include <string>
#include <set>
using namespace std;

struct Par {
  int a, b;

  Par(int A, int B) :
    a(A), b(B) {}

  bool operator<(const Par &p)
  const {
    return a < p.a;
  }
};
```

```
int main() {
  multiset<Par> cj;
  cj.insert(Par(3, 73));
  cj.insert(Par(1, 91));
  cj.insert(Par(2, 52));
  cj.insert(Par(1, 71));
  multiset<Par>::iterator it;
  for (it = cj.begin();
       it != cj.end(); it++) {
    printf("%d\t%d\n", it->a, it->b);
  }
  return 0;
}
```

No vídeo, obtém-se:

```
1	91
1	71
2	52
3	73
```

Iterator com map

```
#include <cstdio>
#include <map>
#include <string>
using namespace std;
int main () {
  map<string,float> nota;
  nota["Pedro"]=10.0;
  nota["Antonio"]=5.0;
  nota["Maria"]=7.5;
  map<string,float>::iterator iter;
  for (iter = nota.begin(); iter != nota.end(); iter++) {
    printf("%s,\t%f\n", iter->first.c_str(), iter->second);
  }
  return 0;
}
```

No vídeo, obtém-se:
Antonio, 5.000000
Maria, 7.500000
Pedro, 10.000000

Iterator com Variáveis auto C++11

```
#include <string>
#include <set>
using namespace std;
int main() {
  set<string> nome;
  nome.insert("joao"); nome.insert("maria"); nome.insert("jose");
  nome.insert("pedro"); nome.insert("maria"); nome.insert("joao");
  for (auto iter = nome.begin(); iter != nome.end(); iter++) {
    printf("%s\n", iter->c_str());
  }
  for (auto iter: nome) {  // iter é uma string, não é iterator.
    printf("%s\n", iter.c_str());
  }
  return 0;
}
```

Iterator com strings

- Os iterators podem ser usados considerando os caracteres separadamente.

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

int main () {
  std::string str ("Test string");
  sort(str.begin(), str.end());
  for (std::string::iterator it = str.begin(); it != str.end(); ++it)
    std::cout << *it;
  std::cout << '\n';

  return 0;
}
```

Exercício Usando iterator e list

- Resolva o problema onlinejudge 11988
- Recomendações
  - Use list<char>.
  - Quando um “home” for encontrado, aponte iterator para inserir no início da lista; quando for um “end”, aponte o iterator para o fim da fila. Não esqueça de atualizar o iterado após cada letra inserida.

Exercício Usando iterator

- Resolva o problema URI 1244 da forma que deseja, mas utilize iterator para apresentar as palavras

Algoritmos

- Algoritmos que podem ser aplicados aos containers através do iterators
- Podem modificar ou não os dados
- Lista em http://www.cplusplus.com/reference/algorithm/

stable_sort

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

struct Ponto {
  float x, y;
  Ponto(float X, float Y) :
    x(X), y(Y) {};
};

void mostra(vector<Ponto> s) {
  vector<Ponto>::iterator it;
  printf("-----\n");
  for (it = s.begin();
       it != s.end(); it++)
  printf("%f\t%f\n", it->x, it->y);
}
```

```
bool operator<(const Ponto &p1,
               const Ponto &p2) {
  return p1.x < p2.x;
}

int main() {
  vector<Ponto> lPt;
  lPt.push_back(Ponto(1.2, 2.3));
  lPt.push_back(Ponto(4.5, 5.6));
  lPt.push_back(Ponto(1.2, 1.2));
  lPt.push_back(Ponto(6.7, 7.8));
  mostra(lPt);
  stable_sort(lPt.begin(), lPt.end());
  mostra(lPt);

  return 0;
}
```

Exercício Usando stable_sort

- Resolva o problema URI 1244 usando vector e stable_sort
- Recomendações
  - Use vector<string> para definir a lista.
  - Use a função stable_sort() que ordenar as palavras.

replace_if

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

bool ehMaior(int i) { return i > 10; }
int main() {
  int vet[] = { 10, 90, 4, -10, 25, 93, 10, 7, 0, 33 };
  for (int i=0; i<10; i++)
    printf("%3d ", vet[i]);
  printf("\n");
  replace_if(vet, vet+10, ehMaior, 10);
  for (int i=0; i<10; i++)
    printf("%3d ", vet[i]);
  return 0;
}
```

No vídeo, obtém-se:

```
 10  90   4 -10  25  93  10   7   0  33
 10  10   4 -10  10  10  10   7   0  10
```

replace_if

```
#include <deque>
#include <algorithm>
#include <string>
using namespace std;

bool iniciaComJ(string s) {
  return s[0] == 'J';
}

int main() {
  deque<string> vet;
  vet.push_back("Joao");
  vet.push_back("Maria");
  vet.push_back("Pedro");
  vet.push_back("Jose");
  vet.push_back("Antonio");
  for (auto i = vet.begin();
       i!=vet.end(); i++)
    printf("%s ", i->c_str());
  printf("\n");
  replace_if(vet.begin(), vet.end(),
                      iniciaComJ, "XXXX");
  for (auto i = vet.begin();
       i!=vet.end(); i++)
    printf("%s ", i->c_str());
  return 0;
}
```

No vídeo, obtém-se:
Joao Maria Pedro Jose Antonio
XXXX Maria Pedro XXXX Antonio

Algoritmos e Iteradores

- Os algoritmos set_difference, set_union, set_intersection, entre outros, realizam operações sobre containers.
- Estas funções recebem iteradores de entrada e saída
- Insert_iterators são iteradores de saída desenvolvidos para permitir que algoritmos que normalmente escrevam elementos possam ser usados para inserir elementos em containers.

```
set_difference(conj1Entra.begin(), conj1Entra.end(),
               conj2Entra.begin(), conj2Entra.end(),
              inserter(conjSaida, conjSaida.begin()));
```

- Veja também: **back_inserter**, **front_inserter** e **insert_iterator**

Algoritmos e Iteradores

- Tipos diferentes de destinos podem ser usados, desde que o insert iterator utilize a rotina de inserção correspondente ao container destino

```
#include <algorithm>
#include <list>
#include <string>
#include <vector>
using namespace std;

int main() {
  vector<string> vet1;
  vet1.push_back("banana");
  vet1.push_back("pera");
  vet1.push_back("maça");
  string vet2[] = {"abacaxi", "laranja", "pera"};
  list<string> l;

  set_difference(vet1.begin(), vet1.end(),
                 vet2, vet2 + 3,
                 front_inserter(l));
   . . .
```

Functores

- Rotinas que retornam instâncias de classes que implementam o operador () usado em comparações nos algoritmos
- A escolha de um functor indica o que ocorrerá no algoritmo escolhido
- Na omissão de um functor, o **less** é normalmente usado

Functores - greater

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

int main() {
  vector<string> vet;
  vet.push_back("Joao");
  vet.push_back("Maria");
  vet.push_back("Pedro");
  vet.push_back("Jose");
  vet.push_back("Antonio");
  sort(vet.begin(), vet.end(),
       greater<string>());
  for (auto i = vet.begin();
       i!=vet.end(); i++)
    printf("%s ", i->c_str());
  printf("\n");
  return 0;
}
```

No vídeo, obtém-se:
Joao Maria Pedro Jose Antonio
XXXX Maria Pedro XXXX Antonio

Containers de containers

- Em containers de containers é necessário inserir os containers mais internos um a um.
- Por exemplo:
  - **`vector<set<int> > adj`**
  - Cria um vetor de conjuntos, mas cada conjunto deve ser inserido explicitamente nos elementos do vector para que possam ser usados