SCC-210 – Algoritmos Avançados

STL: Estruturas de dados lineares e Mapas

João Batista

Standard Template Library - STL

- Biblioteca de classes que utilizam templates para implementar diversas estruturas de dados
- Templates permitem parametrizar a estrutura de dados com um tipo de dados escolhido pelo programador
- Para utilizá-la é necessário um compilador C++ e a biblioteca instalada (*default*)
- É possível utilizar a biblioteca sem ter que desenvolver um código orientado a objetos (C++), veremos vários exemplos sobre isso no curso

Introdução

- STL é uma biblioteca bastante extensa, envolve estruturas de dados básicas e diversos algoritmos tais como:
  - Strings
  - Estruturas de dados básicas e (classes *containers*)
  - Algoritmos de ordenação

- STL contém uma grande quantidade de estruturas de dados básicas (classes *containers*):
  - Vectors
  - Maps
  - Sets
  - Stacks
  - Queues
  - Deques
  - Priority Queues
  - Entre outras...

Introdução

- É importante saber que deve-se tomar diversos cuidados ao se utilizar STL
  - O mais importante é que STL foi projetada tendo-se em mente o desempenho
  - Portanto, os algoritmos estão entre os mais eficientes conhecidos (excelente para maratona!)
  - Entretanto, a biblioteca não oferece muitas verificações de erros cometidos pelo programador

Vectors

- Podem ser considerados vetores dinâmicos
- Permitem a indexação dos elementos, utilizando a mesma sintaxe dos vetores
- O primeiro elemento encontra-se na posição 0
- Para utilizar:

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

- Para declarar:

```
vector<tipo> variável;
```

- Para inserir um valor:

```
push_back(valor);
```

Vectors

- Exemplo:

```
vector<double> exemplo;
exemplo.push_back(1.2);
exemplo.push_back(3.1);
```

- Para os elementos inseridos por push_back é possível acessá-los/modificá-los usando um índice:

```
for (i=0; i < exemplo.size(); i++)
    printf("%f", exemplo[i]);
```

- O método size() fornece o número de exemplos atualmente no vector

Vectors

- Cuidado, nunca acesse um elemento além do tamanho do vetor.

```
vector<int> exemplo;
exemplo[2] = 2;
```

- Todo elemento deve ser primeiro inserido pelo método push_back(), ou

```
vector<char> exemplo(10);    // Cria 10 elems
exemplo[9] = 'A';            // Ok!
```

Iteradores

- Generalização de um ponteiro;
- Permite acessar os elementos de classes *containers* (vector, set, map, etc.).
- São manipulados pelos seguintes operadores sobrecarregados:
  - Incremento e decremento: ++ e --;
  - Igualdade e desigualdade: == e !=;
  - Desreferenciação: *

Iteradores

- Exemplo:

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

int main()
{
    vector<int> vet;
    vector<int>::iterator p;

    for (int i=0; i <=4; i++)
        vet.push_back(i);

    for (p=vet.begin(); p!=vet.end(); p++)
        printf("%d", *p);

    return 0;
}
```

Iteradores

- Cuidado: o método end() retorna um sentinela, e não um iterador para o último elemento
- É necessário declarar:

```
vector<int> vet;
vector<int>::iterator p;
```

- Uma vez que cada classe container implementa o seu iterador

Iteradores

- Existem três categorias de iteradores:
  - Iteradores de avanço: somente ++;
  - Iteradores bidirecionais: ++ e --;
  - Iteradores de acesso aleatório: ++, -- e [].
- Todo iterador de acesso aleatório também é um iterador bidimensional.
- Todo iterador bidirecional também é de avanço.

Iteradores

- Existem duas sub-categorias:
  - Iteradores contantes: não permitem modificar os dados;
  - Iteradores mutáveis: permitem modificar.
- Iteradores constantes servem para trechos de código que não devem alterar os dados.
- Declarados com `const_iterator`.

Iteradores

- Existem também iteradores reversos. Eles resolvem o problema:

```
for (p = o.end(); p != o.begin(); p--)   //Ops...
```

- O método end() retorna um sentinela, e não um iterador para o último elemento.
- Para um reverse iterator o operador ++ caminha para o início do container.

```
reverse_iterator rp;
for (rp = o.rbegin(); rp!= o.rend(); rp++)   // Ok!
```

Map

- Cria associações entre pares de valores. Os elementos do map são mantidos pela ordem da chave. Portanto, é útil quando a ordem das chaves é importante.

- Não permite que duas chaves iguais.

```
#include<map>
map<TChave, TData>
```

Map

- Operações:
  - m.size() – Forneçe o número de elementos no map;
  - m.empty() – Forneçe true se o map estiver vazio;
  - m.insert(E) – Insere E no map;
  - m.erase(C) – Remove o elemento de chave C;
  - m.find(C) – Retorna um iterador para o elemento de chave C, ou m.end() se não existe C;
  - m1==m2 – true se m1 e m2 possuem os mesmos elementos.

Map

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

int main() {
    map<char, int> m;
    map<char, int>::iterator p;

    m.insert(pair<char , int>('c', 2));
    m.insert(make_pair('b', 4));
    m['a'] = 5;
    printf("%d\n", m['b']);
    for (p=m.begin();p!=m.end(); p++)
        printf("%c %d\n", p->first, p->second);
}
```

Map

```
#include<cstdio>
#include<cstring>
#include<map>

using namespace std;

struct ltstr {
    bool operator()(const char* s1, const char* s2) const {
        return strcmp(s1, s2) < 0;
    }
};

int main()
{
    map<const char*, int, ltstr> months;

    months["january"] = 31;
    months["february"] = 28;
    months["march"] = 31;

    printf("june -> %d", months["june"]);
}
```

Filas, Pilhas e Deques

- São estruturas lineares simples.
- Geralmente, não são a solução direta para um problema, mas podem ser úteis como estruturas de dados intermediárias.
- Por exemplo, uma fila é útil para a implementação de uma busca em largura em grafos e uma pilha para uma busca em profundidade.

Deque

- É uma estrutura flexível, similar aos vectors que permitem inserção de elementos em ambas as extremidades.
- Características:
  - Acesso aleatórios aos elementos;
  - Inserção e remoção de elementos em tempo constante nas extremidades;
  - Inserção e remoção de elementos em tempo linear no meio.

Deque

- #include <deque>
- Declaração: deque <T>
- Operações:
  - d.size() – Forneçe o número de elementos na deque;
  - d.empty() – Forneçe true se a deque estiver vazia;
  - d.front() – Forneçe o elemento na frente da deque;
  - d.back() – Forneçe o elemento no final da deque;
  - d.push_front(E) – Insere E no início da deque;
  - d.push_back(E) – Insere E no final da deque;

Deque

- Operações
  - d.pop_front() – Remove o elemento na frente da deque**;
  - d.pop_back() – Remove o elemento no final da deque**;
  - d.insert (I, E) – insere E na posição dada pelo iterador I;
  - d.erase(I) – remove o elemento na posição I;
  - d.clear() – apaga a deque;
  - d1 == d2 – true se d1 e d2 possuem os mesmos elementos na mesma ordem.

Deque - Exemplo

```
#include<cstdio>
#include<deque>

using namespace std;

int main() {
    deque<char> d;
    int i;

    d.push_back('c');
    d.push_front('a');
    d.insert(d.begin() + 1, 'b');
    for (i=0; i<d.size(); i++)
        printf("%c", d[i]);
    printf("%c", d.front());
    printf("%c", d.back());
    d.pop_front();
    d.pop_back();
    printf("%c", d[0]);
    return 0;
}
```

Stacks

- Pilhas
  - #include <stack>
  - Declaração: stack <T>
  - Operações:
    - s.size() – Forneçe o número de elementos na pilha;
    - s.empty() – Forneçe true se a pilha estiver vazia;
    - s.top() – Forneçe o elemento no topo da pilha;
    - s.push(E) – Insere uma cópia de E no topo da pilha;
    - s.pop() – Remove o elemento no topo da pilha**;
    - s1 == s2 – true se s1 e s2 possuem os mesmos elementos.

Stacks - Exemplo

```
#include<cstdio>
#include<stack>

using namespace std;

int main() {
    stack<int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    printf("%d\n", s.top());
    s.pop();
    printf("%d\n", s.top());
}
```

Queues

- Filas
  - #include <queue>
  - Declaração: queue <T>
  - Operações:
    - q.size() – Forneçe o número de elementos na fila;
    - q.empty() – Forneçe true se a fila estiver vazia;
    - q.front() – Forneçe o elemento na frente da fila;
    - q.back() – Forneçe o elemento no final da fila;
    - q.push(E) – Insere E no final da fila;
    - q.pop() – Remove o elemento na frente da fila**;
    - q1 == q2 – true se q1 e q2 possuem os mesmos elementos.

Classes Containers

deque	deque<T>::iterator deque<T>::const_iterator deque<T>::reverse_iterator deque<T>::const_reverse_iterator (iteradores de acesso aleatório)	#include<deque>
stack	Sem iteradores	#include<stack> (adaptador)
queue	Sem iteradores	#include<queue> (adaptador)

Quertyu

PC/Uva IDs: 110301/10082, Popularity: A, Sucess rate: high, Level 1

A common typing error is to place your hands on the keyboard one row to the right of the correct position. Then “Q” is typed as “W” and “J” is typed as “K” and so on. Your task is to decode a message typed in this manner.

Quertyu

Input

Input consists of several lines of text. Each line may contain digits, spaces, uppercase letters (except “Q”, “A”, “Z”), or punctuation shown above [except back-quote (`)]. Keys labeled with words [Tab, BackSp, Control, etc.] are not represented in the input.

Output

You are to replace each letter or punctuation symbol by the one immediately to its left on the QWERTY keyboard shown above. Spaces in the input should be echoed in the output.

Quertyu

Sample Input

O S, GOMR YPFSU/

Sample Output

I AM FINE TODAY.

Exercício: Quertyu

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

using namespace std;

char *str[] = {"`1234567890-=",
               "QWERTYUIOP[]\\",
               "ASDFGHJKL;'",
               "ZXCVBNM,./"};
```

Exercício: Quertyu

```
int main() {
  map<char, char> m;
  int i, j;
  string line;

  for (i=0; i < 4; i++)
    for (j=1; j < strlen(str[i]); j++)
      m[str[i][j]] = str[i][j-1];

  while (getline(cin, line)) {
    for (i=0; i < line.length(); i++)
      if (line[i] == ' ')
        printf("%c", line[i]);
      else
        printf("%c", m[line[i]]);
    printf("\n");
  }
}
```