Laboratório de programação II

Standard Template Library (STL)

Edson Moreno

edson.moreno@pucrs.br

http://www.inf.pucrs.br/~emoreno

Standard Template Library

- Biblioteca padronizada de funções
- Oferece um conjunto de classes de
  - Uso genérico
  - Contêineres (estruturas de dados, como pilhas, listas e filas)
  - Iteradores (objetos que percorrem elementos de um conjunto)
  - Algoritmos básicos (principalmente os destinados a busca e classificação).
- Principais vantagens
  - Armazenam qualquer tipo de dado
  - Simplificação do trabalho com estruturas de dados
    - Código baseado em ponteiros é complexo
  - Possuem um grande conjunto de algoritmos tal como
    - Reverse, swap, sort

Características de STL

- Não usa polimorfismo em função do desempenho
- Usa extensivamente os templates
- Construída basicamente sobre três conceitos
  - Containers
  - Interadores
  - Código genérico

Componentes da STL

- Containers
  - Dividido em três tipos
    - Containers de sequência
    - Containers associativos
    - Containers adaptativos
- Interadores

STL Containers

- Utilizado para organizar um conjunto de objetos de mesmos tipo
- Dividido em três tipos
  - Containers de sequência
    - Permite o armazenamento de diferentes tipos de dados e o acesso sequencial
  - Containers associativos
    - Exploram o conceito de pares chaves e valores, onde cada chave aparece uma única vez
  - Containers adaptativos
    - Adaptam classes previamente definidas nos outros tipos de containers

Container de sequência

- Vetor genérico que armazena coleção de objetos do mesmo tipo
  - Vector
    - Equivalente a um vetor
      - **Vector** tem operações que permitem o seu crescimento e redução de forma dinâmica
  - Tipos de containers de sequencia
    - Vector
    - List
    - deque

Tipos de container de sequência

- **Vector container**
  - Característica
    - Insere objeto sempre no final d lista
  - Equivalente a um vetor
    - **Vector** tem operações que permitem o seu crescimento e redução de forma dinâmica
  - Permite acesso a os seus elemento utilizando índices
    - Índice varia de 0 a n-1, onde n é o tamanho do vetor
      - O acesso pode ser aleatório
  - Para usá-lo
    - #include <vector>

Tipos de container de sequência

- **Vector container**
  - Exemplos de uso de vector container
    - Permite o acesso direto via uso de índice

```
#include <vector>
…
  vector<int> scores (100);                //100 integer scores
  vector<Passenger>passengerList(20);      //list of 20 passengers

   scores[5]=15;
```

Tipos de container de sequência

- **List container**
  - Característica
    - Permite incluir e remover itens em qualquer posição na sequencia
  - Armazena os elementos por posição
  - Cada item na lista tem
    - Valor e endereço de memória (ponteiro)
      - Utilizado para indicar o próximo item na sequência
  - Para acessar um dado na lista
    - Deve-se iniciar pela primeira posição
    - Deve-se seguir de elemento a elemento até localizar o item desejado
  - Não é uma estrutura de acesso direto, como o vector

Tipos de container de sequência

- **List container**
  - Exemplos de uso de list container

```
#include <list>
using namespace std;
…
  list<int> first; // cria lista vazia de inteiros
  list<int> second (4,100); // cria lista com quatro inteiros com valor 100
```

Container associativos

- Característica de funcionamento
  - Funciona com o conceito de chaves
  - Estão sempre ordenados

- Tipos de containers associativos
  - Set: Armazena um conjunto de chaves sem repetição
  - Multiset: Armazena um conjunto de chaves permitindo repetições
  - Map: Armazena um conjunto de pares {chaves, valor} sem repetição
  - Multimap: Armazena um conjunto de pares {chaves, valor} permitindo repetições

Container adaptativos

- Característica de funcionamento
  - Criados a partir da adaptação dos containers de sequencia
  - Utilizam com base o vector, list ou deque

- Tipos de containers adaptativo
  - stack: Funciona como uma pilha (LIFO)
  - queue: Funciona como uma fila (FIFO)
  - Priority_queue: Funciona com uma fila ordenada, sendo o maior valor o primeiro a sair

Métodos comuns aos containers

- ***Operadores*** =, <, >, <=, >=, ==, !=
- ***Construtor padrão:*** possuem conjunto de construtores válidos
- ***Construtor de cópia:*** cria container novo, a partir de um existente
- ***Destrutor:*** destrutor padrão
- ***Empty:*** retorna se o container está vazio
- ***Max_size:*** retorna o número máximo de objetos no container
- ***Size:*** retorna o número de elementos usados
- ***Swap:*** troca todos elementos do container

Métodos específicos dos containers sequenciais

- ***Begin:*** retorna um iterador para o primeiro elemento do container
- ***End:*** retorna iterador para o último elemento do container (posição n, não usada)
- ***Rbegin:*** retorna iterador para o ultimo elemento (posição n-1, usada)
- ***Rend:*** retorna iterador para o elemento anterior ao primero (posição -1, não usada)
- ***Erase:*** apaga um ou mais elementos do container
- ***Clean:*** apaga todos os elementos do container

Navegando nos containers - Iteradores

- Iterador
  - Objeto que pode acessar uma coleção de mesmo tipo
  - Equivalente a um ponteiro, porém no contexto de containers

- Cada tipo de container em STL
  - Possui um iterador próprio com funções apropriadas ao container
    - Um iterador de vector permite acesso aleatório
    - Um iterador de list não pemite acesso aleatório

Tipos de iteradores

- Input
  - Lê um objeto do container, se movendo do início para o fim do container
- Output
  - Escreve um objeto no container, se movendo do início para o fim do container
- Forward
  - Leitura e escrita somente para frente
- Bidirectional
  - Leitura e escrita para frente e para tras
- Random access
  - Leitura e escrita acessando aleatoriamente qualquer objeto do container

Operações com iteradores

- Comum a todos iteradores

*	Retorna o item ao qual o iterador aponta atualmente
++	Move o iteradora para o próximo objeto (pré ou pós incremento)
--	Move o iterador para o objeto anterior (pré e pós decremento)
==	Compara dois iteradores
!=	Compara dois iteradores
=	Atribui um iterador a outro (inválido para o iterador output)

Operações com iteradores

- Específico de random access

p+=i	iterador avança i posições
p-=i	iterador retorna i posições
p+i	retorna iterador avançado i posições de p
p-i	retorna iterador recuando i posições de p
p[i]	retorna referência ao objeto I
p<p1	retorna true se p aponta para objeto anterior a p1
p<=p1	retorna true se p aponta para objeto anterior ou igual a p1
p>p1	retorna true se p aponta para objeto posterior a p1
p>=p1	retorna true se p aponta para objeto posterior ou igual a p1

Containers e iteradores

- Relação entre os tipos de containers e os tipos de interadores
  - vector – Random access iterator
  - deque – Random access iterator
  - list – Bidirectional iterator
  - set, multiset – Bidirectional iterator
  - map, multimap – Bidirectional iterator
  - containers adaptador– Bidirectional iterators

Exemplo de uso do vector

```
01 #include <iostream>
02 #include <vector>
03 #include <string>
04
05 using namespace std;
06
07 main()
08 {
09    vector<string> SS;
10
11    SS.push_back("The number is 10");
12    SS.push_back("The number is 20");
13    SS.push_back("The number is 30");
14
15    cout << "Loop by index:" << endl;
16
17    int ii;
18    for(ii=0; ii < SS.size(); ii++)
19    {
20       cout << SS[ii] << endl;
21    }
22
23    cout << endl << "Constant Iterator:" << endl;
24
25    vector<string>::const_iterator cii;
26    for(cii=SS.begin(); cii!=SS.end(); cii++)
27    {
28       cout << *cii << endl;
29    }
30
31    cout << endl << "Reverse Iterator:" << endl;
32
33    vector<string>::reverse_iterator rii;
34    for(rii=SS.rbegin(); rii!=SS.rend(); ++rii)
35    {
36       cout << *rii << endl;
37    }
38
39    cout << endl << "Sample Output:" << endl;
40
41    cout << SS.size() << endl;
42    cout << SS[2] << endl;
43
44    swap(SS[0], SS[2]);
45    cout << SS[2] << endl;
46 }
```

Exemplo de uso do list

```
01 // Standard Template Library example
02
03 #include <iostream>
04 #include <list>
05 using namespace std;
06
07 // Simple example uses type int
08
09 main()
10 {
11    list<int> L;
12    L.push_back(0);              // Insert a new element at the end
13    L.push_front(0);             // Insert a new element at the beginning
14    L.insert(++L.begin(),2);     // Insert "2" before position of first argument
15                                 // (Place before second argument)
16    L.push_back(5);
17    L.push_back(6);
18
19    list<int>::iterator i;
20
21    for(i=L.begin(); i != L.end(); ++i) cout << *i << " ";
22    cout << endl;
23    return 0;
24 }
```

Exemplo de uso do map

```
// Program: Map Own Class
// Purpose: To demonstrate a map of classes

#include <string>
#include <iostream>
#include <vector>
#include <map>
using namespace std;

class CStudent
{
public :
  int nStudentID;
  int nAge;
public :
  // Default Constructor - Empty
  CStudent()  {  }
  // Full constructor
  CStudent(int nSID, int nA)  {  nStudentID=nSID; nAge=nA;  }
  // Copy constructor
  CStudent(const CStudent& ob)
    {  nStudentID=ob.nStudentID; nAge=ob.nAge;  }
  // Overload =
  void operator = (const CStudent& ob)
    {  nStudentID=ob.nStudentID; nAge=ob.nAge;  }
};
```

```
int main(int argc, char* argv[])
{
  map <string, CStudent> mapStudent;

  mapStudent["Joe Lennon"] = CStudent(103547, 22);
  mapStudent["Phil McCartney"] = CStudent(100723, 22);
  mapStudent["Raoul Starr"] = CStudent(107350, 24);
  mapStudent["Gordon Hamilton"] = CStudent(102330, 22);

  // Access via the name
  cout << "The Student number for Joe Lennon is " <<
    (mapStudent["Joe Lennon"].nStudentID) << endl;

  return 0;
}
```

Exemplo de uso de vector para construção de matriz

```
// Program: Vector of Vectors Demo
// Purpose: To demonstrate nested STL containers

#include <iostream>
#include <vector>

using namespace std;

typedef vector <int> VEC_INT;

int inp[2][2] = {{1, 1}, {2, 0}};
  // Regular 2x2 array to place into the template
int main(int argc, char* argv[])
{
  int i, j;
  vector <VEC_INT> vecvec;
  // if you want to do this in all one step it looks like this
  // vector <vector <int> > vecvec;

  // Fill it in with the array
  VEC_INT v0(inp[0], inp[0]+2);  // passing two pointers
    // for the range of values to be copied to the vector
  VEC_INT v1(inp[1], inp[1]+2);

  vecvec.push_back(v0);
  vecvec.push_back(v1);

  for (i=0; i<2; i++)
  {
    for (j=0; j<2; j++)
    {
      cout << vecvec[i][j] << "  ";
    }
    cout << endl;
  }
  return 0;
}

// Output:
// 1 1
// 2 0
```

Exemplo de map usando swap

```
1  // swap maps
2  #include <iostream>
3  #include <map>
4
5  int main ()
6  {
7    std::map<char,int> foo,bar;
8
9    foo['x']=100;
10   foo['y']=200;
11
12   bar['a']=11;
13   bar['b']=22;
14   bar['c']=33;
15
16   foo.swap(bar);
17
18   std::cout << "foo contains:\n";
19   for (std::map<char,int>::iterator it=foo.begin(); it!=foo.end(); ++it)
20     std::cout << it->first << " => " << it->second << '\n';
21
22   std::cout << "bar contains:\n";
23   for (std::map<char,int>::iterator it=bar.begin(); it!=bar.end(); ++it)
24     std::cout << it->first << " => " << it->second << '\n';
25
26   return 0;
27 }
```

Exemplos online

- Vários exemplos de uso de stl
  - http://www.tenouk.com/cpluscodesnippet/cplusstandardtemplatelibrarystlsindex.html