Escola de Engenharia
Universidade do Minho

Complementos de Programação de Computadores – Aula 7b

Vectors

Mestrado Integrado em Electrónica Industrial e Computadores

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em Reis, Rocha e Faria, 2007)

Escola de Engenharia
Universidade do Minho

Classe vector: Introdução (1)

- **A classe `vector` é uma alternativa à representação de array primitivo**
- **Template de classe**
  - necessário especificar o tipo dos elementos
  - `vector<int> vx;`
- **Necessário incluir o ficheiro "vector.h"**
  - `#include <vector>`
- **Alguns métodos**
  - `vx.size();` *// retorna tamanho do vector vx*
  - `vx.empty();` *// determina se vector vx está vazio*
  - `vx.resize(novo_tamanho);` *// redimensiona vector vx*
  - `vx2=vx;` *// cópia de vectores*

Classe vector: Introdução (2)

- **Definição de um vector com determinado tamanho**
- **Elementos podem ser acedidos através de índice**

```
void copia() {
   const int tam=10;
   vector<int> v1(tam);
   int v2[tam];
   ...
   for (int i=0; i<tam; i++)
       v2[i]=v1[i]
}
```

- **Elementos são inicializados com valor de defeito do tipo. Pode-se atribuir um valor inicial:**
  - `vector<int> v1(10,-1);` *// v1 contém 10 elementos*
    // do tipo inteiro inicializados a -1

Classe vector: Introdução (3)

- **Definição de um vector sem tamanho (vazio)**
  - `vector<int> vy;`
- **Inserir um elemento**
  - `vy.push_back(x);` *// insere x no fim do vector*
  - `vy.pop_back(x2);` *// retira x2 do fim do vector*
- **Uso de iterador**
  - `iterator it;`
  - `it = vy.begin();` *// aponta para 1º elemento*
  - `it = vy.end();` *// aponta para último elemento*
  - `*it` *// elemento do vector referenciado por iterador*
  - `it++` *// incrementa iterador; aponta p/ próximo elemento*

Classe vector: Introdução (4)

```
void teste_vector() {
   const int tam=7;
   vector<int> v1;
   int v2[tam] = {0,1,1,2,3,5,8}; // Inicialização de array
                                  // primitivo
   for (int i=0; i<tam; i++)
       v1.push_back(v2[i]);

   cout << "conteúdo do vector v1 : \n";
   for (vector<int>::iterator it = v1.begin();
                    it != v1.end(); it++)
       { cout << *it << "  "; }   // Valor na posição
                                  // apontada por it
   cout << endl;
}
```


Classe vector : Tamanho e Capacidade

- **Capacidade: Espaço em memória reservado para o objecto**
  - Nº elementos que podem ser colocados no vector sem necessidade de o aumentar
  - Expandida automaticamente, quando necessário
  - `vy.capacity();` *// Capacidade do vector*
  - `vy.reserve(cap);` *// Coloca capacidade do vector igual a cap*
- **Tamanho: Nº de elementos do vector**
  - `vy.size();` *// Tamanho (nº elementos) do vector*

Alocação Dinâmica de Memória

- **Criação dinâmica de objectos : operador `new`**
- **Libertação de memória: operador `delete`**
  - Se não se usar `delete`, espaço ocupado só é libertado no final do programa
  - não há "garbage collection" (ao contrário do que acontece, por exemplo, em Java)
  - `delete` só pode ser usado em objectos que tenham sido criados com `new`
- **Objecto referenciado por mais que um apontador**

```
string *s1 = "bom dia";
string *s2 = s1;
...
delete s2;
```


Arrays Primitivos: Introdução

- **Nome de array é um apontador**
  - `int vp[6]; int vp2[6];`
- **Identificador do array é o endereço 1º elemento**
  - `vp` é `&vp[0]`
- **Conteúdo do array**
  - Conteúdo da 1ª posição: `*vp` ou `vp[0]`
  - Conteúdo da 2ª posição: `*(vp+1)` ou `vp[1]`
- **Passagem de array como parâmetro é por referência**
  - `funcao(int vp[]);` *// Declaração*
  - `funcao(vp);` *// Chamada*
- **Operador atribuição não funciona**
  - vp = vp2; *// errado! Não é possível copiar directamente…*

Arrays Primitivos: Crescimento

```
int *leArray(int & numEls) {
   int tam=0; numEls=0;
   int valorEnt;
   int *arrayN=NULL;
   cout << "Escreva valores inteiros: ";
   while (cin >> valorEnt) {
       if (numEls==tam) {
              int *original = arrayN;
              arrayN = new int[tam*2+1];
              for (int i=0; i<tam; i++)
                     arrayN[i] = original[i];
              delete [] original;
              tam = tam*2+1;
       }
       arrayN[numEls++] = valorEnt;   }
   return arrayN;
}
```


Arrays Primitivos: Crescimento

```
int main() {
   int *v1;
   int n;

   v1 = leArray(n);
   for (int i=0; i<n; i++)
       cout << "v1[" << i  <<"] = " << v1[i] << endl;
   ...
   delete[] v1;
   ...
   return 0;
}
```

`vector` : Aumento da Capacidade

- Redimensiona vector usando método `resize(int)`

```
void leArray(vector<int> & vx) {
   int numEls = 0;
   int valorEnt;
   cout << "Escreva valores inteiros: ";
   while (cin >> valorEnt) {
       if (numEls==vx.size()) {
             vx.resize(vx.size()*2+1);
       vx[numEls++] = valorEnt;
   }
   vx.resize(numEls);
}
```


`vector` : Crescimento Dinâmico

- Dimensão do vector alterada dinamicamente quando se usa push_back

```
void leArray(vector<int> & vx) {
   int valorEnt;
   vx.resize(0);
   cout << "Escreva valores inteiros: ";
   while (cin >> valorEnt)
       vx.push_back(valorEnt);
}
```

vector : Teste de Crescimento Dinâmico

```
int main() {
   vector<int> v1;
   leArray(v1);
   for (int i=0; i<v1.size(); i++)
       cout << "v1[" << i  << "] = " << v1[i] << endl;
   return 0;
}
```


vector: Inserção e Eliminação

- **Inserir em qualquer posição do vector vx**
  - `vx.insert(it, el);` *// Insere `el` na posição referenciada // pelo iterator it*
  - `vx.insert(it, it1OV, it2OV);` *// Insere elementos de // outro vector com inicio em // iterator it1OV e fim it2OV*
- **Eliminar elementos**
  - `vx.erase(it);` *// Elimina elemento referenciado por iterador `it`*
  - `vx.erase(itIni, itFim);` *// Elimina elementos entre os // iteradores `itIni` e `itFim`*

vector: Atribuição, Pesquisa e Troca

- **Atribuição e troca**
  - `vx = vy;` *// elementos de vector vy são copiados para vx*
    // vx é redimensionado
  - `vx.swap(vy);` *// elementos de vx e vy são trocados*

- **Algoritmos genéricos de pesquisa e cópia**
  - `it = find(itIni, itFim, el);`
    // Procura `el` entre iteradores `itIni` e `itFim`
  - `copy(itIni, itFim, it);`
    // Copia elementos entre iteradores `itIni` e `itFim`
    para // a posição `it`.

Nota: ostream_iterator<tipo>(cout,"delimitador");

vector: Exemplo

```
int main() {
   vector<string> frase;
   frase.push_back("olá,");
   frase.push_back("bom");
   frase.push_back("dia");
   frase.push_back("joao");
        // escreve elementos separados por espaço
   copy(frase.begin(),frase.end(),ostream_iterator<string>(cout," "));
   cout << endl;
   cout << "tamanho= " << frase.size(); <<endl;
        // insere elemento "rui" antes de "joao"
   frase.insert(find(frase.begin(),frase.end(),"joao"),"rui");
        // altera último elemento para "antonio"
   frase.back() = "antonio";
   copy(frase.begin(),frase.end(),ostream_iterator<string>(cout," "));
   cout << endl;
   cout << "tamanho= " << frase.size() << endl;
}
```

- Resultado ?

vector: Exemplo (2)

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

int main()
{
   vector<int> meuVector;  // Novo Vector com 0 elementos
   meuVector.push_back(42); // Adicionar elemento 42 no fim do vector
   // Mostrar estatísticas do vector.
   cout << "Tamanho do MeuVector: " << meuVector.size() << endl;
   cout << "Máximo Tamanho do MeuVector: " << meuVector.max_size() << endl;
   cout << "Capacidade de MeuVector: " << meuVector.capacity() << endl;
   // Assegurar que tem espaço para pelo menos 1000 elementos.
   meuVector.reserve(1000);
   cout << endl << "Depois de reservar espaço para 1000 elementos:" << endl;
   cout << "Tamanho do MeuVector: " << meuVector.size() << endl;
   cout << "Máximo Tamanho do MeuVector: " << meuVector.max_size() << endl;
   cout << "Capacidade de MeuVector: " << meuVector.capacity() << endl;
   // Garantir que tem espaço para pelo menos 2000 elementos.
   meuVector.resize(2000);
   cout << endl << "Depois de Resize para 2000 elementos:" << endl;
   cout << "Tamanho do MeuVector: " << meuVector.size() << endl;
   cout << "Máximo Tamanho do MeuVector: " << meuVector.max_size() << endl;
   cout << "Capacidade de MeuVector: " << meuVector.capacity() << endl;
}
```


vector: Exemplo (2)

```
Tamanho do MeuVector: 1
Máximo Tamanho do MeuVector: 1073741823
Capacidade de MeuVector: 1

Depois de reservar espaço para 1000 elementos:

Tamanho do MeuVector: 1
Máximo Tamanho do MeuVector: 1073741823
Capacidade de MeuVector: 1000

Depois de Resize para 2000 elementos:

Tamanho do MeuVector: 2000
Máximo Tamanho do MeuVector: 1073741823
Capacidade de MeuVector: 2000
```

Construtores de Vectores

- #include <vector>
- vector(); *//sem argumentos (por defeito)*
- vector(const vector& c); *// cópia de vector*
- vector(size_type num, const TYPE& val = TYPE());
  // num elementos e valor. Exemplo: vector<int> v1(5, 42);
- vector(input_iterator start, input_iterator end);
  // cria um vector que é inicializado para conter elem entre start e end
- ~vector(); *//destrutor*

Operadores em Vectors

- #include <vector>
- TYPE& operator[](size_type index); *//examina elem individuais*
- const TYPE& operator[](size_type index) const;
- vector operator=(const vector& c2);
- bool operator==(const vector& c1, const vector& c2);
- bool operator!=(const vector& c1, const vector& c2);
  // Vectores são iguais se tamanho é igual e cada membro em cada localização é idêntico
- bool operator<(const vector& c1, const vector& c2);
- bool operator>(const vector& c1, const vector& c2);
- bool operator<=(const vector& c1, const vector& c2);
- bool operator>=(const vector& c1, const vector& c2);

Métodos em Vectores

- assign //assign elements to a vector
- at //returns an element at a specific location
- back //returns a reference to last element of a vector
- begin //returns an iterator to the beginning of the vector
- capacity //returns the number of elements that the vector can hold
- clear //removes all elements from the vector
- empty //true if the vector has no elements
- end //returns an iterator just past the last element of a vector
- erase //removes elements from a vector
- front //returns a reference to the first element of a vector
- insert //inserts elements into the vector

Métodos em Vectores

- max_size //returns the maximum number of elements that the vector can hold
- pop_back //removes the last element of a vector
- push_back //add an element to the end of the vector
- rbegin //returns a reverse_iterator to the end of the vector
- rend //returns a reverse_iterator to the beginning of the vector
- reserve //sets the minimum capacity of the vector
- resize //change the size of the vector
- size //returns the number of items in the vector
- swap //swap the contents of this vector with another

Simple Vector Class

```
#ifndef VECTOR_H
#define VECTOR_H

#include "dsexceptions.h"

template <typename Object>
class Vector
{
  public:
    explicit Vector( int initSize = 0 )
      : theSize( initSize ), theCapacity( initSize + SPARE_CAPACITY )
      { objects = new Object[ theCapacity ]; }
    Vector( const Vector & rhs ) : objects( NULL )
      { operator=( rhs ); }
    ~Vector( )
      { delete [ ] objects; }

    bool empty( ) const
      { return size( ) == 0; }
    int size( ) const
      { return theSize; }
    int capacity( ) const
      { return theCapacity; }
```


Simple Vector Class

```
Object & operator[]( int index ) {
   if( index < 0 || index >= size( ) )
            throw ArrayIndexOutOfBoundsException( );
   return objects[ index ];
}

const Object & operator[]( int index ) const {
   if( index < 0 || index >= size( ) )
            throw ArrayIndexOutOfBoundsException( );
   return objects[ index ];
}

const Vector & operator= ( const Vector & rhs ) {
   if( this != &rhs ) {
        delete [ ] objects;
        theSize = rhs.size( );
        theCapacity = rhs.theCapacity;
        objects = new Object[ capacity( ) ];
        for( int k = 0; k < size( ); k++ )
               objects[ k ] = rhs.objects[ k ];
   }
   return *this;
}
```

Simple Vector Class

```
void resize( int newSize )
 {
       if( newSize > theCapacity )
      reserve( newSize * 2 );
       theSize = newSize;
   }

   void reserve( int newCapacity )
   {
       if( newCapacity < theSize )  return;

       Object *oldArray = objects;

       objects = new Object[ newCapacity ];
       for( int k = 0; k < numToCopy; k++ )
           objects[ k ] = oldArray[ k ];
       theCapacity = newCapacity;
       delete [ ] oldArray;
   }
```


Simple Vector Class

```
// Stacky stuff
    void push_back( const Object & x )
    {
        if( theSize == theCapacity )
            reserve( 2 * theCapacity + 1 );
        objects[ theSize++ ] = x;
    }

    void pop_back( )
    {
        if( empty( ) )
            throw UnderflowException( );
        theSize--;
    }

    const Object & back ( ) const
    {
        if( empty( ) )
            throw UnderflowException( );
        return objects[ theSize - 1 ];
    }
```

Simple Vector Class

```
// Iterator stuff: not bounds checked

    typedef Object * iterator;
    typedef const Object * const_iterator;
    iterator begin( )
      { return &objects[ 0 ]; }
    const_iterator begin( ) const
      { return &objects[ 0 ]; }
    iterator end( )
      { return &objects[ size( ) ]; }
    const_iterator end( ) const
      { return &objects[ size( ) ]; }
    enum { SPARE_CAPACITY = 16 };

 private:
    int theSize;
    int theCapacity;
    Object * objects;
};
#endif
```


Complementos de Programação de Computadores – Aula 7b

Vectors

Mestrado Integrado em Electrónica Industrial e Computadores

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em Reis, Rocha e Faria, 2007)