Vectores em C++

Classe `vector:` Introdução

- 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

- 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

- 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 para próximo elemento*

Classe `vector`: Introdução

```
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 de 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 dinâmicamente 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`.

vector: Exemplo

```
#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

```
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 vector

- vector(); *//sem argumentos (por defeito)*
- vector(const vector& c); *// cópia de vector*
- vector(**size_type** num, const TYPE& val = TYPE());

  // numero de elementos e valor. exemplo: vector<int> v1(5, 42);
- vector(input_iterator start, input_iterator end);

  // cria um vector que é inicializado para conter elementos entre start e end
- ~vector(); *//destrutor*

Operadores em Vectors

- TYPE& operator[](**size_type** index); *//examina elementos 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 igual
- 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*

Consultar descrição da classe `vector` (STL)
(por ex: http://www.cppreference.com/wiki/stl/vector/start)