>>> Programação Orientada a Objetos (POO)

... Bibliotecas Padrões

Prof: André de Freitas Smaira

[~]$ _

>>> Algorithm

[~]$ _

>>> Algorithm

* Algoritmos já implementadas na linguagem C++

[~]$ _

>>> swap

* Troca o conteúdo duas variáveis

```cpp
#include <iostream>     // std::cout
#include <utility>      // std::swap

int main () {

  int x=10, y=20;                   // x:10 y:20
  std::swap(x,y);                   // x:20 y:10

  int foo[4];                       // foo: ?  ?  ?  ?
  int bar[] = {10,20,30,40};        // foo: ?  ?  ?  ?    bar: 10 20 30 40
  std::swap(foo,bar);               // foo: 10 20 30 40   bar: ?  ?  ?  ?

  return 0;
}
```

>>> fill

* Preenche várias posições da memória

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::fill
#include <vector>       // std::vector

int main () {
  std::vector<int> myvector (8);                       // myvector: 0 0 0 0 0 0 0 0

  std::fill (myvector.begin(),myvector.begin()+4,5);   // myvector: 5 5 5 5 0 0 0 0
  std::fill (myvector.begin()+3,myvector.end()-2,8);   // myvector: 5 5 5 8 8 8 0 0

  return 0;
}
```

>>> unique

* Retira elementos consecutivamente repetidos

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::unique, std::distance
#include <vector>       // std::vector

bool myfunction (int i, int j) {
  return (i==j);
}

int main () {
  int myints[] = {10,20,20,20,30,30,20,20,10};           // 10 20 20 20 30 30 20 20 1
  std::vector<int> myvector (myints,myints+9);

  // using default comparison:
  std::vector<int>::iterator it;
  it = std::unique (myvector.begin(), myvector.end());   // 10 20 30 20 10 ?  ?  ?  ?

  myvector.resize( std::distance(myvector.begin(),it) ); // 10 20 30 20 10

  // using predicate comparison:
  std::unique (myvector.begin(), myvector.end(), myfunction);   // (no changes)

  return 0;
}
```

[3. unique]$ _

>>> reverse

* Inverte a ordem dos elementos

```cpp
int main () {
  std::vector<int> myvector;

  for (int i=1; i<10; ++i) myvector.push_back(i);   // 1 2 3 4 5 6 7 8 9
  std::reverse(myvector.begin(),myvector.end());    // 9 8 7 6 5 4 3 2 1
  return 0;
}
```

>>> sort

* Ordena os elementos

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

bool comp (int a,int b) { return (a>b); }
// true se a antes de b, false caso contrário

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8); // 32 71 12 45 26 80 53 33
  std::sort (myvector.begin(), myvector.begin()+4); //(12 32 45 71)26 80 53 33
  std::sort (myvector.begin()+4, myvector.end(), comp); // 12 32 45 71(80 53 33 26)
  return 0;
}
```

>>> stable_sort

* Ordena os elementos mantendo os iguais na mesma ordem

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::stable_sort
#include <vector>       // std::vector

bool compare_as_ints (double i,double j)
{
  return (int(i)<int(j));
}

int main () {
  double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};
  std::vector<double> v1(mydoubles);
  std::stable_sort (v1.begin(), v1.end());
  //1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67
  std::vector<double> v2(mydoubles);
  std::stable_sort (v2.begin(), v2.end(), compare_as_ints);
  //1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
  return 0;
}
```

>>> next_permutation

* Próxima permutação

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::next_permutation, std::sort

int main () {
  int myints[] = {3,2,1};

  std::cout << "The 3! possible permutations with 3 elements:\n";
  while ( std::next_permutation(myints,myints+3) )
    std::cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';
  std::cout << "After loop: " << myints[0] << ' ' << myints[1] << ' ' << myints[2] <<
  return 0;
}
```

>>> is_sorted

* Verifica se ordenados

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::is_sorted, std::prev_permutation
#include <array>        // std::array

int main () {
  std::vector<int> foo = {2,4,1,3};

  while (not std::is_sorted(foo.begin(),foo.end())) {
    // try a new permutation:
    std::prev_permutation(foo.begin(),foo.end());
  }
  std::cout << "the range is sorted!\n";
  return 0;
}
```

>>> binary_search

* Busca binária

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::binary_search, std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i>j); }

int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  std::vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3 2 1
  std::sort (v.begin(), v.end()); // 1 1 2 2 3 3 4 4 5
  std::binary_search (v.begin(), v.end(), 3); //true
  std::sort (v.begin(), v.end(), myfunction); // 5 4 4 3 3 2 2 1 1
  std::binary_search (v.begin(), v.end(), 6, myfunction); //false
  return 0;
}
```

>>> min,max

* Encontra um elemento

```
#include <iostream>     // std::cout
#include <algorithm>    // std::min

bool comp(int i, int j) { return i>j; }

int main () {
  int myints[] = {3,7,2,5,6,4,9};

  std::min(1,2); //1
  std::min('a','z'); //'a'
  std::min(3.14,2.72); //2.72
  *std::min_element(myints,myints+7); //2
  *std::min_element(myints,myints+7,comp); //9

  std::max(1,2); //2
  std::max('a','z'); //'z'
  std::max(3.14,2.72); //3.14
  *std::max_element(myints,myints+7);//9
  *std::max_element(myints,myints+7,comp); //2
  return 0;
}
```

>>> find

* Mínimo/máximo entre elementos

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::find
#include <vector>       // std::vector

int main () {
  int myints[] = { 10, 20, 30, 40 };
  int *p;

  p = std::find (myints, myints+4, 30);
  if(p != myints+4)
    std::cout << "Element found in myints: " << *p << '\n';
  else
    std::cout << "Element not found in myints\n";

  std::vector<int> myvector (myints,myints+4);
  std::vector<int>::iterator it;

  it = find (myvector.begin(), myvector.end(), 30);
  if(it != myvector.end())
    std::cout << "Element found in myvector: " << *it << '\n';
  else
    std::cout << "Element not found in myvector\n";

  return 0;
}
```

>>> remove

* Desloca para o fim do vetor, todas as ocorrências

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::remove

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};      // 10 20 30 30 20 10 10 20

  // bounds of range:
  int* pbegin = myints;                          // ^
  int* pend = myints+sizeof(myints)/sizeof(int); // ^                       ^

  pend = std::remove (pbegin, pend, 20);         // 10 30 30 10 10 ?  ?  ?
                                                 // ^              ^
  std::cout << "range contains:";
  for (int* p=pbegin; p!=pend; ++p)
    std::cout << ' ' << *p;
  std::cout << '\n';

  return 0;
}
```

>>> remove_if

* Como o **remove**, mas com determinada condição

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::remove_if

bool IsOdd (int i) { return ((i%2)==1); }

int main () {
  int myints[] = {1,2,3,4,5,6,7,8,9};            // 1 2 3 4 5 6 7 8 9

  // bounds of range:
  int* pbegin = myints;                          // ^
  int* pend = myints+sizeof(myints)/sizeof(int); // ^                 ^

  pend = std::remove_if (pbegin, pend, IsOdd);   // 2 4 6 8 ? ? ? ? ?
                                                 // ^       ^
  std::cout << "the range contains:";
  for (int* p=pbegin; p!=pend; ++p)
    std::cout << ' ' << *p;
  std::cout << '\n';

  return 0;
}
```

>>> transform

* Modifica todos os elementos

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::transform
#include <vector>       // std::vector
#include <functional>   // std::plus

int op_increase (int i) { return ++i; }

int main () {
  std::vector<int> foo;
  std::vector<int> bar;

  for (int i=1; i<6; i++)
    foo.push_back (i*10);                          // foo: 10 20 30 40 50

  bar.resize(foo.size());                          // allocate space

  std::transform (foo.begin(), foo.end(), bar.begin(), op_increase);
                                                   // bar: 11 21 31 41 51

    std::transform (foo.begin(), foo.end(), bar.begin(), foo.begin(), std::plus<int>());
                                                   // foo: 21 41 61 81 101

  std::cout << "foo contains:";
  for (std::vector<int>::iterator it=foo.begin(); it!=foo.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

>>> partition

* Separa elementos conforme uma condição

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::transform
#include <vector>       // std::vector
#include <functional>   // std::plus

int op_increase (int i) { return ++i; }

int main () {
  std::vector<int> foo;
  std::vector<int> bar;

  for (int i=1; i<6; i++)
    foo.push_back (i*10);                         // foo: 10 20 30 40 50

  bar.resize(foo.size());                         // allocate space

  std::transform (foo.begin(), foo.end(), bar.begin(), op_increase);
                                                  // bar: 11 21 31 41 51

    std::transform (foo.begin(), foo.end(), bar.begin(), foo.begin(), std::plus<int>());
                                                  // foo: 21 41 61 81 101

  std::cout << "foo contains:";
  for (std::vector<int>::iterator it=foo.begin(); it!=foo.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

>>> copy

* Copia os elementos de um vetor

```cpp
#include <iostream>     // std::cout
#include <algorithm>    // std::copy
#include <vector>       // std::vector

int main () {
  int myints[]={10,20,30,40,50,60,70};
  std::vector<int> myvector (7);

  std::copy ( myints, myints+7, myvector.begin() );

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it = myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}
```

>>> Functional

>>> Functors

* std::plus: soma de dois valores
* std::minus: subtração de dois valores
* std::multiplies: multiplicação de dois valores
* std::divides: divisão de dois valores
* std::modulus: módulo (resto da divisão)
* std::negate: negação de um valor
* std::equal_to: verifica se dois valores são iguais
* std::not_equal_to: verifica se dois valores são diferentes
* std::greater, std::greater_equal, std::less, std::less_equal: operadores de comparação

>>> Funções que alteram funções

* **std::not_fn**: nega o resultado de uma função

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

bool is_even(int n) {
    return n % 2 == 0;
}

int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5, 6};

    auto is_odd = std::not_fn(is_even);

    std::vector<int> odd_numbers;
    std::copy_if(numbers.begin(), numbers.end(),
                 std::back_inserter(odd_numbers), is_odd);
    return 0;
}
```

>>> Funções que alteram funções

* **std::bind**: fixa parâmetros de uma função

```cpp
#include <iostream>
#include <functional>

void print_sum(int a, int b) {
    std::cout << "A soma é: " << (a + b) << std::endl;
}

int main() {
    auto add_ten = std::bind(print_sum, 10, std::placeholders::_1);
    add_ten(5); // Isso imprime: A soma é: 15
    add_ten(20); // Isso imprime: A soma é: 30
    return 0;
}
```

>>> Numeric

>>> accumulate

* Resultado ao acumular os elementos do vetor

```
#include <iostream>     // std::cout
#include <functional>   // std::minus
#include <numeric>      // std::accumulate

int myfunction (int x, int y) {return x+2*y;}
struct myclass {
^^Iint operator()(int x, int y) {return x+3*y;}
} myobject;

int main () {
  int init = 100;
  int numbers[] = {10,20,30};
  std::cout << "using default accumulate: ";
  std::cout << std::accumulate(numbers,numbers+3,init) << '\n';
  std::cout << "using functional's minus: ";
  std::cout << std::accumulate (numbers, numbers+3, init, std::minus<int>()) << '\n';
  std::cout << "using custom function: ";
  std::cout << std::accumulate (numbers, numbers+3, init, myfunction) << '\n';
  std::cout << "using custom object: ";
  std::cout << std::accumulate (numbers, numbers+3, init, myobject) << '\n';
  return 0;
}
```

>>> inner_product

* Produto escalar

```cpp
#include <iostream>     // std::cout
#include <functional>   // std::minus, std::divides
#include <numeric>      // std::inner_product

int myaccumulator (int x, int y) {return x-y;}
int myproduct (int x, int y) {return x+y;}

int main () {
  int init = 100;
  int series1[] = {10,20,30};
  int series2[] = {1,2,3};

  std::cout << "using default inner_product: ";
  std::cout << std::inner_product(series1,series1+3,series2,init);
  std::cout << '\n';

  std::cout << "using functional operations: ";
  std::cout << std::inner_product(series1,series1+3,series2,init,
                                  std::minus<int>(),std::divides<int>());
  std::cout << '\n';

  std::cout << "using custom functions: ";
  std::cout << std::inner_product(series1,series1+3,series2,init,
                                  myaccumulator,myproduct);
  std::cout << '\n';

  return 0;
}
```

>>> partial_sum

* Soma parcial

```cpp
#include <iostream>     // std::cout
#include <functional>   // std::multiplies
#include <numeric>      // std::partial_sum

int myop (int x, int y) {return x+y+1;}

int main () {
  int val[] = {1,2,3,4,5};
  int result[5];
  std::partial_sum (val, val+5, result);
  std::cout << "using default partial_sum: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  std::partial_sum (val, val+5, result, std::multiplies<int>());
  std::cout << "using functional operation multiplies: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  std::partial_sum (val, val+5, result, myop);
  std::cout << "using custom function: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  return 0;
}
```

>>> adjacent_difference

* Diferenças entre os elementos

```cpp
#include <iostream>     // std::cout
#include <functional>   // std::multiplies
#include <numeric>      // std::adjacent_difference

int myop (int x, int y) {return x+y;}

int main () {
  int val[] = {1,2,3,5,9,11,12};
  int result[7];
  std::adjacent_difference (val, val+7, result);
  std::cout << "using default adjacent_difference: ";
  for (int i=0; i<7; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  std::adjacent_difference (val, val+7, result, std::multiplies<int>());
  std::cout << "using functional operation multiplies: ";
  for (int i=0; i<7; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  std::adjacent_difference (val, val+7, result, myop);
  std::cout << "using custom function: ";
  for (int i=0; i<7; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
  return 0;
}
```

* Aulas do grupo Maratona IFSC (Eu e Ian Giesta)
* http://www.cplusplus.com/