# *Standard Template Library (STL) II*

## BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

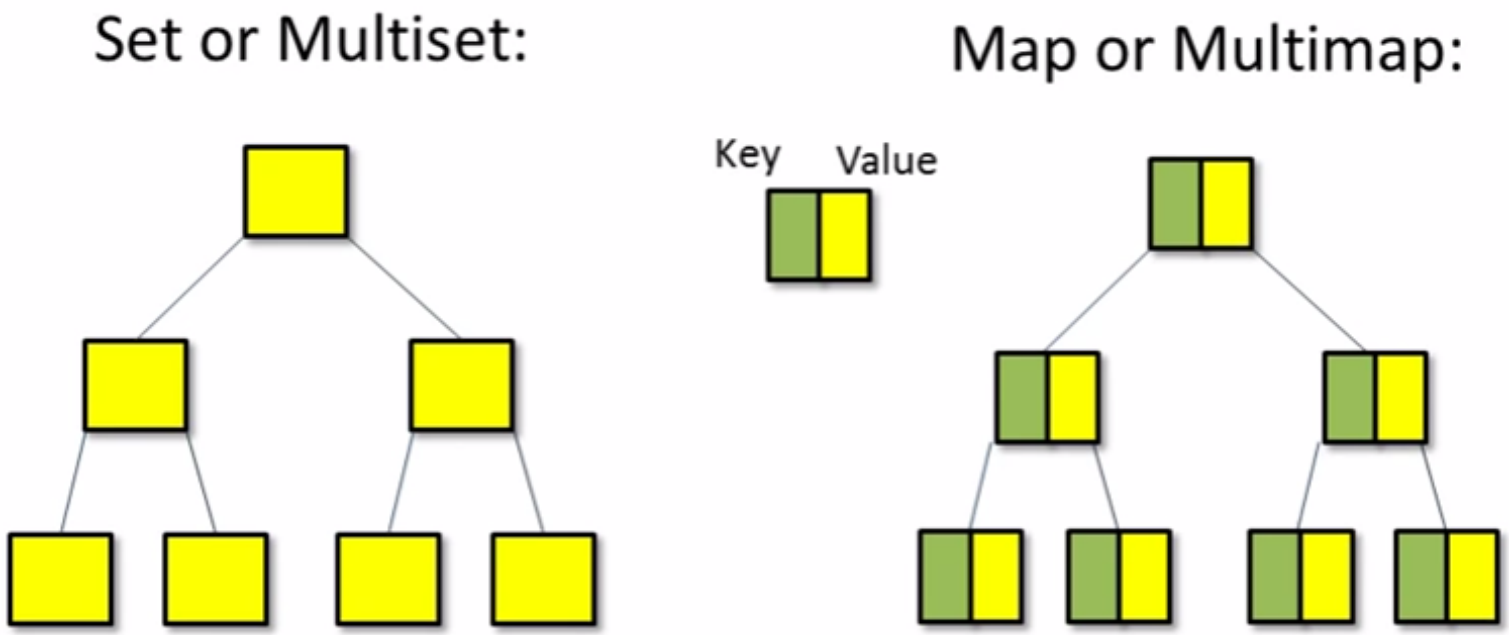

# Contêineres Associativos

- Acesso direto para armazenar/recuperar elementos
- Usa chaves
  - Realiza busca por chaves
- Quatro tipos: **multiset, set, multimap** e **map**
  - Ordenados por chaves

# Contêineres Associativos (cont.)

- **multiset** e **set** manipulam conjunto de valores
  - Valores são próprias chaves
- **multimap** e **map** manipulam valores associados com chaves
  - Possuem chaves e valores
- **multimap** e **multiset** permitem chaves duplicadas enquanto **set** e **map** não permitem

# Contêineres Associativos (cont.)



# Contêiner Associativo *multiset*

- Cabeçalho: #include<set>
- Armazenamento rápido, recuperação de chaves (sem valores)
- Permite duplicatas
- Iteradores bidirecionais

# Contêiner Associativo *multiset* (cont.)

- Ordenação de elementos
  - Feito por objeto com função comparadora (`operator<`)
- Usado na criação do *multiset*
  - Para *multiset* inteiro
    - less <int> objeto com função comparadora
    - multiset < int , std :: less <int> > myObject;
    - Elementos são armazenados em ordem crescente

# Contêiner Associativo *multiset* (cont.)

- Funções multiset
  - ms. insert (value)
    - Insere valor no *multiset*
  - ms.count(value)
    - Retorna número de ocorrências do *value*
  - ms.find (value)
    - Retorna iterador para primeira ocorrência do *value*

# Contêiner Associativo *multiset* (cont.)

- ms.lower_bound(value)
  - Retorna iterador para a primeira ocorrência do *value*
- ms.upper_bound(value)
  - Retorna iterador da primeira posição depois da última ocorrência do *value*

# Contêiner Associativo *multiset* (cont.)

- Classe *pair*
  - Manipula pares de valores
  - Objeto *pair* contem *first* e *second*
    - const_iterators
  - Para um objeto pair q
    - q = ms.equal_range(value)
    - Ajusta *first* e *second* para lower_bound e upper_bound para um dado *value*

# Contêiner Associativo *multiset* (cont.)

```
#include <set> // definição da template de classe multiset
#include <algorithm> // algoritmo copy
#include <iterator> // ostream_iterator
using namespace std;

// define o nome abreviado para o tipo multiset utilizado nesse programa
typedef std::multiset< int, std::less< int > > lms;

int main()
{
    const int SIZE = 10;
    int a[SIZE] = {7,22,9,1,18,30,100,22,85,13};
    lms intMultiset; // lms é o typedef para multiset integer
    std::ostream_iterator< int > output(cout, " ");

    cout<<"There are currently "<<intMultiset.count(15)
        << " values of 15 in the multiset\n";  // 0
```

# Contêiner Associativo *multiset* (cont.)

```
intMultiset.insert(15); // insere 15 em intMultiset
intMultiset.insert(15); // insere 15 em intMultiset
cout << "After inserts, there are "
     << intMultiset.count(15)  // 2
     << " values of 15 in the multiset\n\n";

// iterador que não pode ser utilizado para alterar valores de elemento
lms::const_iterator result;

// localiza 15 em intMultiset; find retorna iterador
// retorna end() se o elemento não é encontrado
result = intMultiset.find(15);

if (result != intMultiset.end()) // se o iterador não
    estiver no fim
     cout << "Found value 15\n"; // localizou valor de
         pesquisa 15
```

# Contêiner Associativo *multiset* (cont.)

```
// localiza 20 em intMultiset; find retorna iterador
result = intMultiset.find(20);

if (result == intMultiset.end()) // será true daí porque
    cout << "Did not find value 20\n"; // não localizou
        20

// insere elementos do array a em intMultiset
intMultiset.insert(a, a + SIZE);
cout << "\nAfter insert, intMultiset contains:\n";
std::copy(intMultiset.begin(), intMultiset.end(),
    output);
//for (auto itr = intMultiset.begin(); itr != intMultiset.end(); ++itr)
// cout << itr << ;
```

# Contêiner Associativo *multiset* (cont.)

```
    // determina limite inferior e superior de 22 em intMultiset
    // intMultiset:{1,7,9,13,15,15,18,22,22,30,85,100}
    cout << "\n\nLower bound of 22: "
         << *(intMultiset.lower_bound(22)); // 22
    cout << "\nUpper bound of 22: "
         << *(intMultiset.upper_bound(22)); // 30

    // p representa par de const_iterators
    std::pair< lms::const_iterator, lms::const_iterator
        > p;

    // usa equal_range para determinar limite inferior e superior
    // de 22 em intMultiset
    p = intMultiset.equal_range(22);
    cout << "\n\nequal_range of 22:"
         << "\n    Lower bound: " << *(p.first)
         << "\n    Upper bound: " << *(p.second);
    cout << endl;
    return 0;
} // fim de main
```

# Contêiner Associativo *set*

- Cabeçalho: #include<set>
- Implementação idêntica de *multiset*
- Chaves únicas
  - Duplicatas ignoradas e não inseridas
- Suporta iteradores bidirecionais
  - Não aceita acesso aleatório
- std :: set< type, std :: less <type>> name;

# Contêiner Associativo *set* (cont.)

```
#include <iostream>
#include <iterator>
#include <string>
#include <set>
#include <algorithm>
#include <functional>

using namespace std;

struct Point { double x, y; };
struct PointCmp {
    bool operator()(const Point& lhs, const Point& rhs)
        const {
        return std::hypot(lhs.x, lhs.y) < std::hypot(
            rhs.x, rhs.y);
    }
};
```

# Contêiner Associativo *set* (cont.)

```
int main()
{
    // (1) Constructor default
    set<string> a;
    a.insert("cat");
    a.insert("dog");
    a.insert("horse");
    for (auto& elem : a) // cat, dog, horse
        cout << elem << " ";
    cout << endl;
    // (2) Constructor iterador
    set<string> b(a.find("dog"), a.end());
    for (auto& elem : b) // dog, horse
        cout << elem << " ";
    cout << endl;
```

# Contêiner Associativo *set* (cont.)

```
// (3) Copia por constructor
set<string> c(a);
c.insert("another horse");
for (auto& elem : c) // another horse, cat, dog, horse
    cout << elem << " ";
cout << endl;

// (4) Move constructor
set<string> d(std::move(a));
for (auto& elem : d) // cat, dog, horse
    cout << elem << " ";
cout << endl;
std::cout << "moved-from set is: ";
for (auto& elem : a) // Vazio
    cout << elem << " ";
cout << endl;
```

# Contêiner Associativo *set* (cont.)

```
    // (5) Inicializa lista constructor
    set<string> e{ "one", "two", "three", "five", "
        eight" };
    for (auto& elem : e) // eight, five, one, three, two
        cout << elem << " ";
    cout << endl;

    // comparação de Points
    set<Point, PointCmp> z = { { 2, 5 }, { 3, 4 }, { 1,
        1 } };
    z.insert({ 1, -1 }); // falha porque a magnitude de (1,-1) é
        igual a (1,1)
    for (auto& elem : z) // (1,1), (3,4), (2,5)
        cout << "(" << elem.x << "," << elem.y << ") ";
    cout << endl;
    return 0;
}
```

# Container Associativo *multimap*

- Cabeçalho #include<map>
- **Armazenamento rápido** e recuperação de chaves e valores associados
  - Tem pares chave/valor
- Chaves duplicadas são permitidas (múltiplos valores para uma única chave)
  - Relação um-para-muitos
  - Ou seja, um estudante pode fazer múltiplos cursos

# Container Associativo *multimap* (cont.)

- Insere objetos *pair* (com uma chave e valor)
- Iteradores bidirecionais

```
std::multimap< int, double, std::less< int > >
    mmapObject;
```

- Tipo de chave *int* e tipo de valor *double*
- Ordenados em ordem crescente das chaves

# Container Associativo *multimap* (cont.)

- Use *typedef* para simplificar o código

```
typedef std::multimap<int, double, std::less<int>>
    mmid;
mmid mmapObject;
mmapObject.insert(mmid::value_type(1, 3.4));
```

- Insere chave 1 com valor 3.4

- mmid::value_type cria um objeto *pair*

# Container Associativo *multimap* (cont.)

- Operações:
  - Inserção: $O(\log_2 n)$
  - Remoção: $O(\log_2 n)$
  - Busca: $O(\log_2 n)$

# Container Associativo *multimap* (cont.)

```
#include <iostream>
#include <map> // definição da template de classe map
using namespace std;

// define o nome abreviado para o tipo multimap utilizado nesse programa
typedef std::multimap< int, double,
                       std::less< int > > Mmid;

int main()
{
    Mmid pairs; // declara os pares de multimap

    cout << "There are currently " << pairs.count(15)
         << " pairs with key 15 in the multimap\n"; //0

    // insere dois objetos value_type em pares
    pairs.insert(Mmid::value_type(15, 2.7));
    pairs.insert(Mmid::value_type(15, 99.3));
```

# Container Associativo *multimap* (cont.)

```
cout << "After inserts, there are "
     << pairs.count(15) // 2
     << " pairs with key 15\n\n";

// insere cinco objetos value_type em pares
pairs.insert(Mmid::value_type(30, 111.11));
pairs.insert(Mmid::value_type(10, 22.22));
pairs.insert(Mmid::value_type(25, 33.333));
pairs.insert(Mmid::value_type(20, 9.345));
pairs.insert(Mmid::value_type(5, 77.54));
```

# Container Associativo *multimap* (cont.)

```
    cout << "Multimap pairs contains:\nKey\tValue\n";

    // utiliza const_iterator para percorrer elementos de pares
    for (Mmid::const_iterator iter = pairs.begin();
         iter != pairs.end(); ++iter)
         cout << iter->first << '\t' << iter->second <<
             '\n';

    cout << endl;
    return 0;
} // fim de main
```

# Container Associativo *multimap* (cont.)

```
There are currently 0 pairs with key 15 in the multimap
After inserts, there are 2 pairs with key 15

Multimap pairs contains:
Key      Value
5        77.54
10       22.22
15       2.7
15       99.3
20       9.345
25       33.333
30       111.11
```

# Container Associativo *map*

- Cabeçalho #include<map>
- Como *multimap*, mas somente pares chave/valor únicos
  - Mapeamento um-para-um (duplicatas ignoradas)
- Usa [] para acessar valores
- Ex.: para objeto map m

```
m[30] = 4000.21;
```

  - Ajusta o valor da chave 30 para 4000.21
- Declaração de tipo

```
std::map< int, double, std::less< int > >;
```

# Container Associativo *map* (cont.)

```
#include <iostream>
#include <map>
#include <string>
#include <utility>
using namespace std;

int main()
{
    typedef map<string, int> mapType;
    mapType popMap;

    popMap.insert(pair<string, int>("China", 1339));
    popMap.insert(pair<string, int>("India", 1187));
    popMap.insert(mapType::value_type("US", 310));
   popMap.insert(mapType::value_type("Indonesia",234));
    popMap.insert(make_pair("Brasil", 193));
    popMap.insert(make_pair("Pakistan", 170));
    popMap["Italia"] = 60;
    popMap["Alemanha"] = 81;
```

# Container Associativo *map* (cont.)

```
// Erase the end element using the erase function
// Because it's ordered map (by key),
// map elements are not in the order of the entry
// In this map it's US since it's ordered alphabetically.
mapType::iterator iter = --popMap.end();
popMap.erase(iter);

// output the size of the map
cout << "Size of populationMap: " << popMap.size()
    << '\n';  //
    7

for (iter = popMap.begin(); iter != popMap.end();
    ++iter) {
     cout << iter->first << ": "
          << iter->second << " million\n";
     }
```

# Container Associativo *map* (cont.)

```
    // find will return an iterator to the matching element if it is found
    // or to the end of the map if the key is not found
    string country("Indonesia");
    iter = popnMap.find(country);
    if (iter != popMap.end())
        cout << country << "'s populations is "//234
             << iter->second << " million\n";
    else
        cout << "Key is not in populationMap" << '\n';

    // clear the entries in the map
    popMap.clear();
        return 0;
}
```

# Algoritmos

- A STL inclui aproximadamente 85 algoritmos
- Podem ser **utilizados genericamente**, em **vários** tipos de **contêineres**.
- Os algoritmos **operam indiretamente** sobre os elementos de um **contêiner usando iteradores**
  - **Vários** deles **utilizam pares de iteradores**, um apontando para o **início** e outro apontando para o **final**;

# Algoritmos (cont.)

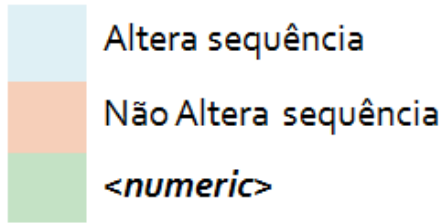
- Os algoritmos **operam indiretamente** sobre os elementos de um **contêiner usando iteradores**
  - **Frequentemente os algoritmos** também **retornam iteradores** como resultado;
  - **Este desacoplamento** dos contêineres **permite** que os **algoritmos sejam genéricos**.

# Algoritmos (cont.)

- A STL **é extensível**
  - Ou seja, é **possível adicionar novos algoritmos** a ela.
- Entre os vários algoritmos disponíveis, temos:
  - Algoritmos não alteradores de sequências;
  - Algoritmos alteradores de sequências;
  - Algoritmos de ordenação
  - Algoritmos numéricos
    - Definidos em <numeric>.

# Algoritmos (cont.)

| | | |
|---|---|---|
| *copy* | *remove* | *reverse_copy* |
| *copy_backward* | *remove_copy* | *rotate* |
| *fill* | *remove_copy_if* | *rotate_copy* |
| *fill_n* | *remove_if* | *stable_partition* |
| *generate* | *replace* | *swap* |
| *generate_n* | *replace_copy* | *swap_ranges* |
| *iter_swap* | *replace_copy_if* | *transform* |
| *partition* | *replace_if* | *unique* |
| *random_shuffle* | *reverse* | *unique_copy* |




| | | |
|---|---|---|
| *adjacent_find* | *find* | *find_if* |
| *count* | *find_each* | *mismatch* |
| *count_if* | *find_end* | *search* |
| *equal* | *find_first_of* | *search_n* |

| | |
|---|---|
| *accumulate* | *partial_sum* |
| *inner_product* | *adjacent_difference* |

# Algoritmos não alteradores de sequências

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

typedef vector<int>::iterator itrVec;

bool menor(int x){ return x < 10; }

int main()
{
    vector<int> vec = { 9, 60, 90, 8, 45, 87, 90, 69,
        69, 55, 7 };
    itrVec it;
    pair<itrVec, itrVec> pair_itr;
```

# Algoritmos não alteradores de sequências (cont.)

```
//1. contando
int n = count(vec.begin(), vec.end(), 69); // 2
int m = count_if(vec.begin(), vec.end(),
                 [](int x){return x < 10; }); // 3
int m2 = count_if(vec.begin(), vec.end(),
                  menor); // 3

//2. Min and Max
it = max_element(vec.begin()+3, vec.end()); // 90 a
    partir pos 3
it = max_element(vec.begin(), vec.end(), [](int x,
    int y){return x %10 < y %10; });
pair_itr = minmax_element(vec.begin(), vec.end());
// primeiro menor, último maior
cout << *pair_itr.first << " " << *pair_itr.second
    << endl;
```

# Algoritmos não alteradores de sequências (cont.)

```
//3. Busca linear (usado quando a data não esta ordenada)
// retorna o primeiro
it = find(vec.begin(), vec.end(), 55);
if (it != vec.end()) cout << "\nEncontrado";
else cout << "\nNao encontrado";

it = find_if(vec.begin(), vec.end(), [](int x){
    return x > 50; }); //60
cout << "\n" << *it;

// encontra dois 69 consecutivos
it = search_n(vec.begin(), vec.end(), 2, 69);
cout << "\n" << *it;
```

# Algoritmos não alteradores de sequências (cont.)

```
// procura algum
vector<int> item = {87, 69, 2};
    it = find_first_of(vec.begin(), vec.end(),
                       item.begin(), item.end());
cout << "\n" << *it;   // 87
it = find_first_of(vec.begin(), vec.end(),
                   item.begin(), item.end(),
                   [](int x, int y){return x == 4 *
                        y; });
cout << "\n" << *it; // 8

// dois elementos iguais consecutivos
it = adjacent_find(vec.begin(), vec.end());
cout << "\n" << *it; // 69
```

# Algoritmos não alteradores de sequências (cont.)

```
    //4. Comparação
    vector<int> vec2(vec);
    if (equal(vec.begin(), vec.end(), vec2.begin()))
        cout << "\nvec e vec são iguais";

    //5. Verificando atributos
    if ( is_sorted(vec.begin(), vec.end()) )
        cout << "\nOrdenado";
    it = is_sorted_until(vec.begin(), vec.end());   // 8
    cout << "\nParcialmente ordenado até " << *it;
        return 0;
}
```

# Algoritmos alteradores de sequências

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

int main(){
    vector<int> vec = {9,60,70,8,45,87,90}; //7 items
    vector<int> vec2(11,0); // 11 items

    //1. Copiar (tamanho destinho >= que fonte)
    copy(vec.begin(), vec.end(), // source
         vec2.begin() );         // destination
    vec2.assign(11,0);
    copy_if(vec.begin(), vec.end(), // source
            vec2.begin(),           // destination
            [](int x){return x > 80; });    // condition
    // vec2: 87,90,0,0,0,0,0,0,0,0,0
```

# Algoritmos alteradores de sequências (cont.)

```
vec2.assign(11, 0);
copy_n(vec.begin(), 4, vec2.begin());
// vec2: 9,60,70,8,0,0,0,0,0,0,0

vec2.assign(11, 0);
copy_backward(vec.begin(), vec.end(), // source
     vec2.end());                     // destination);
// vec2: 0,0,0,0,9,60,70,8,45,87,90
```

# Algoritmos alteradores de sequências (cont.)

```
//2. Transformar
vec2 = vec;
vector<int> vec3(11, 0);
transform(vec.begin(), vec.end(), // source
     vec3.begin(),   //destination
     [](int x){return x - 1; });   // operation
     // vec3: 8,59,69,7,44,86,89,0,0,0,0

     transform(vec.begin(), vec.end(), // source1
               vec2.begin(), //source2
               vec3.begin(), //destination
     [](int x, int y){return x + y; }); //operation
// vec3: 18,120,140,16,90,174,180,0,0,0,0
```

# Algoritmos alteradores de sequências (cont.)

```
// 3. swap
swap_ranges(vec.begin(), vec.end(), // source
      vec3.begin()); //destination
// vec: 18,120,140,16,90,174,180
      // vec3: 9,60,70,8,87,90,0,0,0,0

// 4. Fill
fill(vec.begin(), vec.end(), 9);
// vec1 : 9,9,9,9,9,9,9
vec.assign(7, 0);
fill_n(vec.begin(), 3, 9);
generate(vec.begin(), vec.end(), rand);
vec.assign(7, 0);
generate_n(vec.begin(), 3, rand);
```

# Algoritmos alteradores de sequências (cont.)

```
    // 5. Replace
    vec2.assign({9, 60, 70, 8, 45, 87, 90});
    replace(vec2.begin(), vec2.end(), // intervalo
        9, // valor antigo
        6); // novo valor
    // vec2 : 6,60,70,8,45,87,90
    replace_if(vec2.begin(), vec2.end(), // intervalo
        [](int x){return x > 80; }, // valor antigo
        100); // novo valor
    // vec2 : 6,60,70,8,45,100,100
    return 0;
}
```

# Algoritmo de ordenação

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <functional>
using namespace std;

int main()
{
    vector<int> vec = { 9, 1, 10, 2, 45, 3, 90, 4, 9,
        5, 8 };
    ostream_iterator<int> out(cout, " ");

    sort(vec.begin(), vec.end());
    // vec : 1,2,3,4,5,8,9,9,10,45,90

    sort(vec.begin(), vec.end(), [](int x, int y){
        return x %10 < y %10; });
    // vec : 10,90,1,2,3,4,45,5,8,9,9
```

# Algoritmo de ordenação (cont.)

```
partial_sort(vec.begin(), vec.begin()+5, vec.end(),
     greater<int>());
// vec : 90,45,10,9,9,1,2,3,4,5,8

partial_sort(vec.begin(), vec.begin() + 5, vec.end
    ());
// vec : 1,2,3,4,5,90,45,10,9,9,8

nth_element(vec.begin(),
        vec.begin() + vec.size() / 2,
        vec.end()); // 8
cout << "A mediana é "
     << vec[vec.size() / 2] << '\n';

nth_element(vec.begin(),
        vec.begin() + 1,
        vec.end(),
        greater<int>());// 45
cout << "O segundo maior elemento: "
     << vec[1] << '\n';
```

# Algoritmo de ordenação (cont.)

```
    // Mover todos os valores menores a 10 para o frente do vetor
    vec.assign({ 9, 1, 10, 2, 45, 3, 90, 4, 9, 5, 8 });
    partition(vec.begin(), vec.end(),
              [](int x){return x < 10; });
    // vec : 9,1,8,2,5,3,9,4,90,45,10

    // preserva a ordem original dentro de cada partição
    vec.assign({ 9, 1, 10, 2, 45, 3, 90, 4, 9, 5, 8 });
    stable_partition(vec.begin(), vec.end(),
              [](int x){return x < 10; });
    // vec : 9,1,2,3,4,9,5,8,10,45,90

    //Heapsort algorithm
    make_heap(vec.begin(), vec.end());
    sort_heap(vec.begin(), vec.end());
    // vec : 1,2,3,4,5,8,9,9,10,45,90
        return 0;
}
```

# Exemplo - Campeonato

```
class Jogador{
    string nome, sobrenome;
    int idade;
public:
    Jogador(string nom="", string snom="", int idade=0)
        :
        nome(nom), sobrenome(snom), idade(idade){};
    string getNome()const {return nome; }
    friend ostream& operator<<(ostream& out, const
        Jogador& obj){
       out << endl << obj.nome <<" " << obj.sobrenome
          << " "<< obj.idade;
        return out;
    }
};
```

# Exemplo - Campeonato (cont.)

```
class Time{
    map<int, Jogador> v;
public:
    Time(){};
    void insereJogador(int camiseta, Jogador j){
        //v.insert(pair<int, Jogador>(camiseta, j));
        v[camiseta] = j;
    }
    friend ostream& operator<<(ostream& out,
                               const Time& obj){
        // map<int, Jogador>::const_iterator ou auto
        for (auto item = obj.v.begin();
                  item != obj.v.end(); item++){
            out << item->first << " "
                << item->second << endl;
        }
        return out;
    }
};
```

# Exemplo - Campeonato (cont.)

```
class Campeonato{
    map<string, Time> t;
public:
    Campeonato(){}
    void insereTime(string nome, Time time){
        //v.insert(pair<int, Jogador>(camiseta, j));
        t[nome] = time;
    }
    friend ostream& operator<<(ostream& out, const
        Campeonato& obj){
        //map<string, Time>::const_iterator ou auto
        for (auto item = obj.t.begin();
                item != obj.t.end(); item++){
            out << item->first << endl
                << item->second << endl;
        }
        return out;
    }
};
```

# Exemplo - Campeonato (cont.)

```
int main(){
    Time time1, time2;
    Jogador j1("Mario", "Dias", 23);
    Jogador j2("Alberto", "Soares", 20);
    Jogador j3("Sandro", "Silva", 33);
    time1.insereJogador(1, j1);
    time1.insereJogador(10,j2);
    time1.insereJogador(4, j3);
    Jogador j4("Joao", "Dias", 31);
    Jogador j5("Jose", "Soares", 30);
    Jogador j6("Luiz", "Silva", 19);
    time2.insereJogador(10, j4);
    time2.insereJogador(5, j5);
    time2.insereJogador(3, j6);

    Campeonato camp;
    camp.insereTime("Tabajara", time1);
    camp.insereTime("Naosei", time2);
    cout << camp;
}
```

# Exemplo - Campeonato (cont.)

```
Naosei
3 Luiz Silva 19
5 Jose Soares 30
10 Joao Dias 31


Tabajara
1 Mario Dias 23
4 Sandro Silva 33
10 Alberto Soares 20
```

# Exemplo - Campeonato com ponteiros inteligentes

```
class Time{
    map<int, shared_ptr<Jogador>> v;
public:
    Time(){};
    void insereJogador(int camiseta, shared_ptr<Jogador
        > j){
        //v.insert(pair¡int, Jogador¿(camiseta, j));
        v[camiseta] = j;
    }
    friend ostream& operator<<(ostream& out, const Time
        & obj){
    // map<int, shared_ptr<Jogador>>::const_iterator
        for (auto item = obj.v.begin();
                  item != obj.v.end(); item++){
            out << item->first << " "<< *(item->second)
                << endl;
        }
        return out;
    }
};
```

# Exemplo - Campeonato com ponteiros inteligentes (cont.)

```
class Campeonato{
    map<string, shared_ptr<Time>> t;
public:
    Campeonato(){}
    void insereTime(string nome, shared_ptr<Time> time)
        {
        //v.insert(pair¡int, Jogador¿(camiseta, j));
        t[nome] = time;
    }
    friend ostream& operator<<(ostream& out, const
        Campeonato& obj){
        //map<string, shared_ptr<Time>>::const_iterator
        for (auto item = obj.t.begin();
                item != obj.t.end(); item++){
            out << (item->first) << endl << *(item->
                second) << endl;
        }
        return out;
    }
};
```

# Exemplo - Campeonato com ponteiros inteligentes (cont.)

```
int main(){
    shared_ptr<Time> time1 = make_shared<Time>();
    time1->insereJogador(1,
        make_shared<Jogador>("Mario", "Dias", 23));
    time1->insereJogador(10,
        make_shared<Jogador>("Alberto", "Soares", 20));
    time1->insereJogador(4,
        make_shared<Jogador>("Sandra", "Silva", 33));
    shared_ptr<Time> time2 = make_shared<Time>();
    time2->insereJogador(10,
        make_shared<Jogador>("Joao", "Dias", 31));
    time2->insereJogador(5,
        make_shared<Jogador>("Jose", "Soares", 30));
    time2->insereJogador(3,
        make_shared<Jogador>("Luiz", "Silva", 19));
    Campeonato camp;
    camp.insereTime("Tabajara", time1);
    camp.insereTime("Naosei", time2);
    cout << camp;
}
```

FIM