>>> Programação Orientada a Objetos (POO)

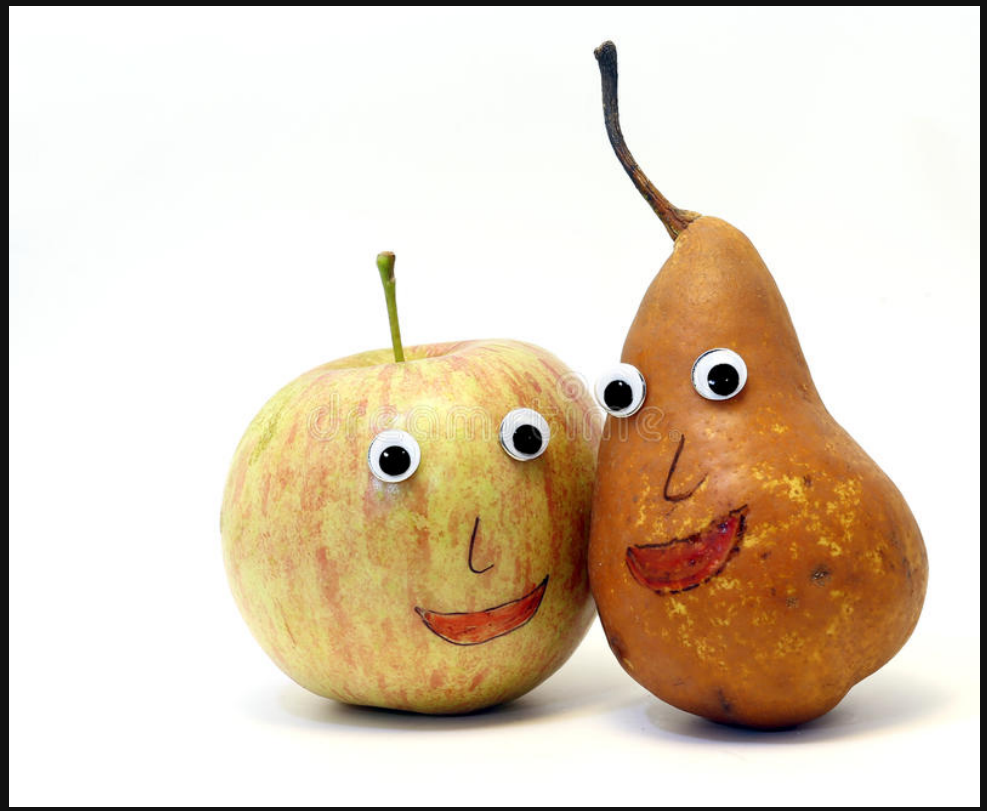
... Standard Template Library (STL)

Prof: André de Freitas Smaira

[~]$ _

>>> Par - pair

* Par



>>> pair - Declaração e acesso

```cpp
#include <utility>      // std::pair, std::make_pair
#include <string>       // std::string
#include <iostream>     // std::cout
using namespace std;

int main () {
  pair <string,double> p1;                        //
  pair <string,double> p2 ("tomate",2.30);   // inicialização
  pair <string,double> p3 (p2);              // copia p2
  p1 = make_pair(string("lampadas"),0.99);   // make_pair
  p2.first = "sapatos";                 // ("sapatos",2.30)
  p2.second = 39.90;                    // ("sapatos",39.90)
  return 0;
}
```

* Por padrão, ordenado pelo primeiro elemento
* http://www.cplusplus.com/reference/utility/pair/

>>> Template de dois tipos?

```
template <typename T1, typename T2>
class Par {
private:
    T1 primeiro;
    T2 segundo;

public:
    Par(T1 p, T2 s) : primeiro(p), segundo(s) {}

    void mostrar() {
        cout << "Primeiro: " << primeiro
             << ", Segundo: " << segundo << endl;
    }

    T1 getPrimeiro() {
        return primeiro;
    }

    T2 getSegundo() {
        return segundo;
    }
};
```

>>> Vetores - vector

* **Ponteiros**: coisas chatas e perigosas...
* Já pensou se existisse uma estrutura que **não precisasse ser pré-alocada**... e que ainda **não precisasse** se preocupar em **alocar dinamicamente**?
* Sim... **Ela existe**... **vector**
* cplusplus.com: "Vectors are sequence containers representing arrays that can change in size."

>>> vector - Declaração e iterador

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

int main ()
{
  // construtores
  vector<int> primeiro;                                // vector de int vazio
  vector<int> segundo (4,100);                       // 4 inteiros com valor 100
  vector<int> terceiro (second.begin(),second.end());  // copia do segundo
  vector<int> quarto (terceiro);                       // copia do terceiro

  // tambem pode copiar array de C
  int myints[] = {16,2,77,29};
  vector<int> quinto (myints, myints + sizeof(myints) / sizeof(int) );

  printf("quinto:");
  for (vector<int>::iterator it = quinto.begin(); it != quinto.end(); ++it)
    printf(" %d",*it);
  printf("\n"); //quinto: 16 2 77 29
  return 0;
}
```

>>> vector - Inclusão, tamanho e acesso

```cpp
int main()
{
    vector<int> v;
    for(int i=0; i<10; i++)
        v.push_back(i);     // adiciona i ao fim de v
    for(int i=0; i<v.size(); i++)
        printf("%d ",v[i]); // imprime v
    return 0;
}
```

>>> vector - Outras coisas

```cpp
int main()
{
    vector<int> v = {1,2,3,4,5};
    v.erase(v.begin()+3);
    for(int i=0; i<v.size(); i++)
        printf("%d ",v[i]); printf("\n"); //1 2 3 5
    v.erase(v.begin(),v.begin()+2);
    for(auto it=v.begin(); it!=v.end(); it++) //c++11
        printf("%d ",*it); printf("\n"); //3 5
    v.insert(v.begin(),2,1);
    for(auto e:v) //c++11
        printf("%d ",e); printf("\n"); //1 1 3 5
    v.clear();
    printf("%d\n",v.size()); //0
    return 0;
}// g++ -std=c++11 prog.cpp -o prog
```

* http://www.cplusplus.com/reference/vector/vector/

>>> vector - principais métodos

* begin() e end(): iteradores para o começo e fim
* size(): número de elementos armazenados
* empty(): se está vazio
* front(): primeiro elemento
* back(): último elemento
* push_back(): insere na última posição
* pop_back(): tira o último elemento
* insert(): insere elementos
* erase(): apaga elementos
* clear(): remove todos os elementos

>>> Exercício

Mude esse programa (lisra01 ex5) para usar **vector**

```
#include<iostream>
int main() {
    int lista[100000];
    int n = 0;
    while(std::cin >> lista[n]) {
        int found = 0;
        for(int i=0; i<n; i++)
            if(lista[i] == lista[n]) {
                found = 1;
                for(int j=i+1; j<=n-1; j++)
                    lista[j-1] = lista[j];
                n--;
                break;
            }
        if(found == 0) n++;
    }
    if(n == 0) lista[0] = 0;
    std::cout << lista[0] << std::endl;
    return 0;
}
```

>>> Exercício

```cpp
#include<iostream>
#include<vector>
int main() {
    vector<int> lista;
    int num;
    while(std::cin >> num) {
        lista.push_back(num);
        for(auto it = lista.begin(); it != lista.end()-1; ++it)
            if(*it == num) {
                lista.erase(it);
                lista.pop_back();
                break;
            }
    }
    if(lista.size() == 0) lista.push_back(0);
    std::cout << lista[0] << std::endl;
    return 0;
}
```

>>> Cadeia de caracteres - string

* array (C) está para vector (C++) assim como char[] (C) está para **string** (C++)

>>> string - Declaração

```cpp
#include <stdio.h>        // printf
#include <iostream>        // std::cout
#include <string>        // std::string
using namespace std;

int main ()
{
  string s0 ("String inicial"); // String inicial

  string s1;    //
  string s2 (s0); // String inicial
  string s3 (s0, 8, 3); // nic
  string s4 ("A character sequence"); //A character sequence
  string s5 ("Another character sequence", 12);//Another char
  string s6a (10, 'x');         // xxxxxxxxxx
  string s6b (10, 42);       // **********
  string s7 (s0.begin(), s0.begin()+6); //String
  string s8 = "C string"; // C string
  string s9 = '!'; // !
  string s10 = s8+s9; // C string!

  cout << s0 << endl; //String inicial
  printf("%s\n",s0.c_str()); //String inicial
  return 0;
}
```

>>> string - Tamanho e acesso

```cpp
#include <stdio.h>
#include <iostream>
#include <string>

int main ()
{
  std::string str ("Test string");
  for ( int i=0; i<str.size(); i++)
    printf("%c",str[i]);//Test string
  printf("\n");
  for ( auto it=str.begin(); it!=str.end(); it++)
    printf("%c",*it);//Test string
  printf("\n");
  return 0;
}
```

>>> string - Outras coisas

```cpp
#include <iostream>        // std::cout
#include <string>          // std::string
using namespace std;

int main ()
{
  string s1 = "Alpha";
  string s2 = "Beta";
  if(s1<s2) cout << s1 << " vem antes de " << s2 << "\n";
  else if(s1>s2) cout << s1 << " vem depois de " << s2 << "\n"
  else cout << s1 << " eh igual a " << s2 << "\n";
  return 0;
}
```

* `scanf("%[^\n]",s);` está para `char[]` assim como `getline()` está para `string`
* http://www.cplusplus.com/reference/string/string/

>>> string - principais métodos

* begin() e end(): iteradores para o começo e fim
* size(): número de elementos armazenados
* empty(): se está vazio
* front(): primeiro elemento
* back(): último elemento
* push_back(): insere na última posição
* pop_back(): tira o último elemento
* insert(): insere elementos
* erase(): apaga elementos
* replace(): substitui parte da string
* clear(): remove todos os elementos
* find(): encontra parte da string
* substr(): retorna parte da string

>>> Strings - Regex

* Encontrando coisas…

```
string texto = "Eu tenho 31 anos";
cout << texto.find('31') << endl; // 8
cout << texto.substr(pos, 2) << endl; // 31
```

* E se eu não souber qual o número?

```
numero = "";
for(char c : texto)
    if(isdigit(c))
        numero += c;
cout << numero << endl;
```

* E se houver **mais de um número** e eu quiser **somente o primeiro**?

```
string numero = "";
bool achei = false;
for(char c : texto)
    if(isdigit(c)) {
        achei = true;
        numero += c;
    } else if(achei)
        break;
cout << numero << endl;
```

>>> Strings - Regex

* E se houver **mais de um número e eu quiser todos**?

```
string numero = "";
vector<string> numeros;
bool achei = false;
for(char c : texto)
    if(isdigit(c)) {
        achei = true;
        numero += c;
    } else if(achei) {
        achei = False
        numeros.append(numero)
        numero = ''
    }
for(string num : numeros)
    cout << num << endl;
```

>>> Strings - Regex

* Não tem **jeito mais fácil**? **CLARO!**
* Se eu **não souber qual o número**:

```
smatch match;
regex_search(texto, match, regex(R"(\d+)"));
cout << match.str(0) << endl;
```

* E se eu **não souber se existe** o número:

```
smatch match;
if(regex_search(texto, match, regex(R"(\d+)")))
    cout << match.str(0) << endl;
```

* E se houver mais de um número e eu quiser **somente o primeiro**?

```
smatch match;
if(regex_search(texto, match, regex(R"(\d+)")))
    cout << match.str(0) << endl;
```

>>> Strings - Regex

* E se houver mais de um número e eu **quiser todos**?

```
regex reg(R"(\d+)");
sregex_iterator currentMatch(texto.begin(), texto.end(), reg);
sregex_iterator endMatch;
for(; currentMatch != endMatch; ++currentMatch)
    cout << currentMatch->str() << endl;
```

* E se quiser as **posições**?

```
regex reg(R"(\d+)");
sregex_iterator currentMatch(texto.begin(), texto.end(), reg);
sregex_iterator endMatch;
for(; currentMatch != endMatch; ++currentMatch) {
    int i = currentMatch->position();
    int j = i+currentMatch->length();
    cout << "texto[" << i << ":" << j << "] = "
            << currentMatch->str() << endl;
}
```

>>> Strings - Regex

* Quantificadores:
  * Pelo menos um dígito:
    ```
    regex_search(texto, match, regex(R"(\d+)"))
    ```
  * Qualquer quantidade de dígitos:
    ```
    regex_search(texto, match, regex(R"(\d*)"))
    ```
  * Exatamente 2 dígitos:
    ```
    regex_search(texto, match, regex(R"(\d{2})"))
    ```
  * De 1 a 2 dígitos:
    ```
    regex_search(texto, match, regex(R"(\d{1,2})"))
    ```
  * 0 ou 1 dígito:
    ```
    regex_search(texto, match, regex(R"(\d?)"))
    ```
  * Dígito no início:
    ```
    regex_search(texto, match, regex(R"(^\d)"))
    ```
  * Dígito no fim:
    ```
    regex_search(texto, match, regex(R"(\d$)"))
    ```

>>> Strings - Regex

* Tipos de caracteres:
  * Qualquer dígito:
    regex_search(texto, match, regex(R"(\d)"))
  * Qualquer não dígito:
    regex_search(texto, match, regex(R"(\D)"))
  * Qualquer dígito, letra ou _:
    regex_search(texto, match, regex(R"(\w)"))
  * O contrário do anterior:
    regex_search(texto, match, regex(R"(\W)"))
  * Qualquer espaço:
    regex_search(texto, match, regex(R"(\s)"))
  * Qualquer não espaço:
    regex_search(texto, match, regex(R"(\S)"))
  * Qualquer de alguns caracteres:
    regex_search(texto, match, regex(R"([abc])"))
  * Qualquer exceto alguns caracteres:
    regex_search(texto, match, regex(R"([^abc])"))
  * Qualquer de caracteres em um intervalo:
    regex_search(texto, match, regex(R"([a-c])")) // ASCII

>>> Strings - Regex

* Tipos de caracteres:
    * Qualquer de algumas palavras:
      ```
      regex_search(texto, match, regex(R"(um|dois)"))
      ```
    * Escape de caracteres especiais:
      ```
      regex_search(texto, match, regex(R"([\(\)\[\]])"))
      // Qualquer um entre os caracteres (, ), [, ]
      ```
    * Qualquer caractere diferente de '\n':
      ```
      regex_search(texto, match, regex(R"(.)"))
      ```
    * Grupos:
      ```
      regex_search(texto, match,
                   regex(R"Tenho (\d+) gatos e (\d+) cachorros"))
      ```

>>> Strings - Regex

Outras utilidades:

* Substituição

```
regex_replace(texto, regex(R"(\d+)"), "");
```

* Separação

```
regex reg(R"(\s)");
sregex_token_iterator iter(texto.begin(), texto.end(), reg, -1);
sregex_token_iterator fim;
vector<std::string> resultado(iter, fim);
```

>>> Strings - Regex

* Qualquer combinação dentre os citados (exemplos):
  * Lista de numeros inteiros:
    ```
    regex_search(texto, match, regex(R"(\[[\d,\-\s]+\])"))
    ```
  * Comentários de uma linha em um código de c++:
    ```
    regex_search(texto, match, regex(R"(//.*)"))
    ```
* Um pouco mais sobre:
  https://www.studyplan.dev/pro-cpp/regex
* Site para testes: https://regex101.com/

>>> Mas regex não vimos...

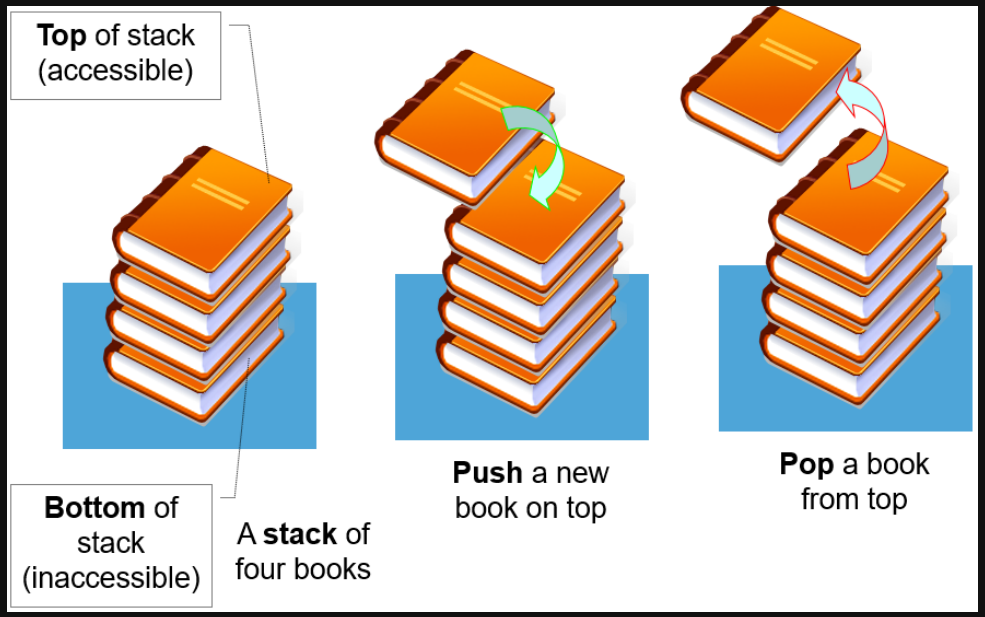
Faça uma função que receba uma frase no formato "Sou "André", tenho 34 anos e moro em São Carlos-SP". O nome, a idade, a cidade e o estao vão mudar. Sua função deve retornar uma instância da seguinte struct com esses dados preenchidos:

```
#include<string>
#include<regex>
using namespace std;
struct Pessoa {
    string nome, cidade, estado;
    int idade;
};

Pessoa dados(string str) {
    Pessoa pessoa;
    regex reg(R"(^Sou \"([^\"]+)\", tenho (\d+) anos e moro em (.+)-([A-Z]{2})$)");
    smatch match;
    if(regex_search(str, match, reg)) {
        pessoa.nome = match[1];
        pessoa.idade = stoi(match[2].str());
        pessoa.cidade = match[3];
        pessoa.estado = match[4];
    }
    return pessoa;
}
```

>>> Pilha - stack

* Pilha de livros



* Operações básicas - GIF
* cplusplus.com: "Stacks (...) operate in LIFO (last-in first-out), where elements are inserted and extracted only from one end (...)."
* STACK NÃO tem iterador

>>> stack - Declaração

```cpp
#include <iostream>        // std::cout
#include <stack>           // std::stack
#include <vector>          // std::vector
using namespace std;

int main ()
{
  vector<int> v(2,200);         // vector com 2 elementos 200

  stack<int> s1;                      // stack vazia

  stack<int,vector<int> > s2;  // stack vazia implementada em vector
  stack<int,vector<int> > s3(v); // stack copia de v

  cout << s1.size() << '\n'; // 0
  cout << s2.size() << '\n'; // 0
  cout << s3.size() << '\n'; // 2

  return 0;
}
```

>>> stack - Inserção, remoção, acesso e verificação

```cpp
#include <stdio.h>        // printf
#include <stack>          // std::stack

int main ()
{
  std::stack<int> s;
  for(int i=0; i<5; ++i)
    s.push(i);    // insere de 0 a 4

  while (not s.empty()) // Enquanto nao vazia
  {
     s.top() += s.size();    // soma o tamanho da pilha ao topo
     printf(" %d",s.top()); // imprime o topo atualizado
     s.pop();    // retira o topo (ATENCAO: sem retorno)
  }
  printf("\nTamanho: %d\n",s.size());//0
  return 0;
}
```

* http://www.cplusplus.com/reference/stack/stack/

>>> stack - principais métodos

* size(): número de elementos armazenados
* empty(): se está vazio
* push: insere um elemento
* pop(): tira um elemento
* top(): retorna o elemento do topo

>>> Pilha (stack) - Exercício

Faça uma função `bool testa_parenteses(std::string expressao)` que receba uma string `expressao` e use a `stack` para verificar se o balanceamento de parênteses está correto, isto é, dada uma expressão matemática que contenha parênteses, verifique se para cada parênteses fechando, tem um abrindo antes e vice-e-versa. Retorne `true` se estiver correto e `false` caso contrário.

>>> Pilha (stack) - Exercício

```
bool testa_parenteses(char *expressao) {
    stack<char> p;
    for(char c : expressao) {
        if(c == '(')
            p.push(c);
        else if(c == ')') {
            if(p.empty())
                return false;
            p.pop();
        }
    }
    return p.empty();
}
```

>>> Fila - queue

* Fila da biblioteca fora da lei (não preferencial)

* Operações básicas - GIF
* cplusplus.com: "Queues (...) operate in FIFO (first-in first-out), where elements are inserted into one end of the container and extracted from the other."
* QUEUE NÃO tem iterador

>>> queue - Declaração

```cpp
#include <stdio.h>        // printf
#include <vector>          // std::vector
#include <queue>           // std::queue
using namespace std;

int main ()
{
  vector<int> v (2,200);          // 2 elementos 200

  queue<int> q1;                  // fila vazia

  queue<int,vector<int> > q2; // fila vazia com implementacao em vec
  queue<int,vector<int> > q3 (v); // copia de v

  printf("%d\n",q1.size());// 0
  printf("%d\n",q2.size());// 0
  printf("%d\n",q3.size());// 2

  return 0;
}
```

[5. Fila - queue]$ _

>>> queue - Inserção, remoção, acesso e verificação

```cpp
#include <iostream>       // std::cout
#include <queue>          // std::queue
using namespace std;

int main ()
{
  queue<int> q;
  int sum = 0;

  for (int i=1;i<=10;i++)
     q.push(i);

  while (not q.empty())
  {
     sum += q.front();
     q.pop();
  }

  cout << sum << '\n';//55

  return 0;
}
```

* http://www.cplusplus.com/reference/queue/queue/

>>> queue - principais métodos

* size(): número de elementos armazenados
* empty(): se está vazio
* push: insere um elemento
* pop(): tira um elemento
* front(): retorna o elemento da frente
* back(): retorna o elemento de trás

>>> Exercício

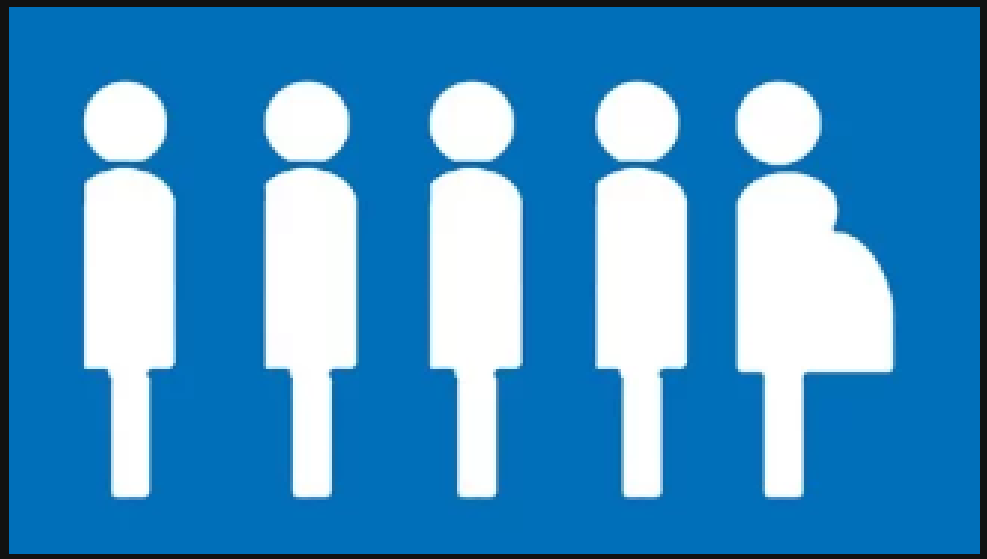
Faça uma função que recebe uma queue **fila** de nomes de clientes e um vector **atendimento** também de nomes. **atendimento** indica a ordem em que as pessoas que estavam na **fila** foram atendidas. Retorne **true** se elas foram atendidas na ordem correta e **false** caso contrário.

>>> Exercício

```cpp
#include<string>
#include<queue>
using namespace std;
bool verifica(queue<string> fila, vector<string> atendimento) {
    for(string nome : atendimento) {
        if(nome != fila.front())
            return false;
        fila.pop();
    }
    return true;
}
```

>>> Fila de prioridades - priority_queue

* Fila dentro da lei



* cplusplus.com: "Priority queues are (...) designed such that its first element is always the greatest of the elements it contains (...)."
* PRIORITY QUEUE NÃO tem iterador

>>> priority_queue - Declaração

```cpp
#include <iostream>       // std::cout
#include <queue>          // std::priority_queue
#include <vector>         // std::vector
#include <functional>     // std::greater
using namespace std;

int main ()
{
  int myints[]= {10,60,50,20};

  priority_queue<int> pq1; // vazia
  priority_queue<int> pq2 (myints,myints+4); // elementos de myints
  priority_queue<int, vector<int>, greater<int> > pq3 (myints,myints+4);
  // implementacao em vector com ordenacao invertida

  while(not pq2.empty()) { cout << " " << pq2.top(); pq2.pop(); } printf("\n");
  // 60 50 20 10
  while(not pq3.empty()) { cout << " " << pq3.top(); pq3.pop(); } printf("\n");
  // 10 20 50 60

  return 0;
}
```

>>> priority_queue - Inserção, remoção, acesso e verificação

```cpp
#include <iostream>       // std::cout
#include <queue>          // std::priority_queue

int main ()
{
  std::priority_queue<int> pq;
  int sum =0;

  for (int i=1;i<=10;i++) pq.push(i);

  while (not pq.empty())
  {
     sum += pq.top();
     pq.pop();
  }

  std::cout << "total: " << sum << '\n';

  return 0;
}
```

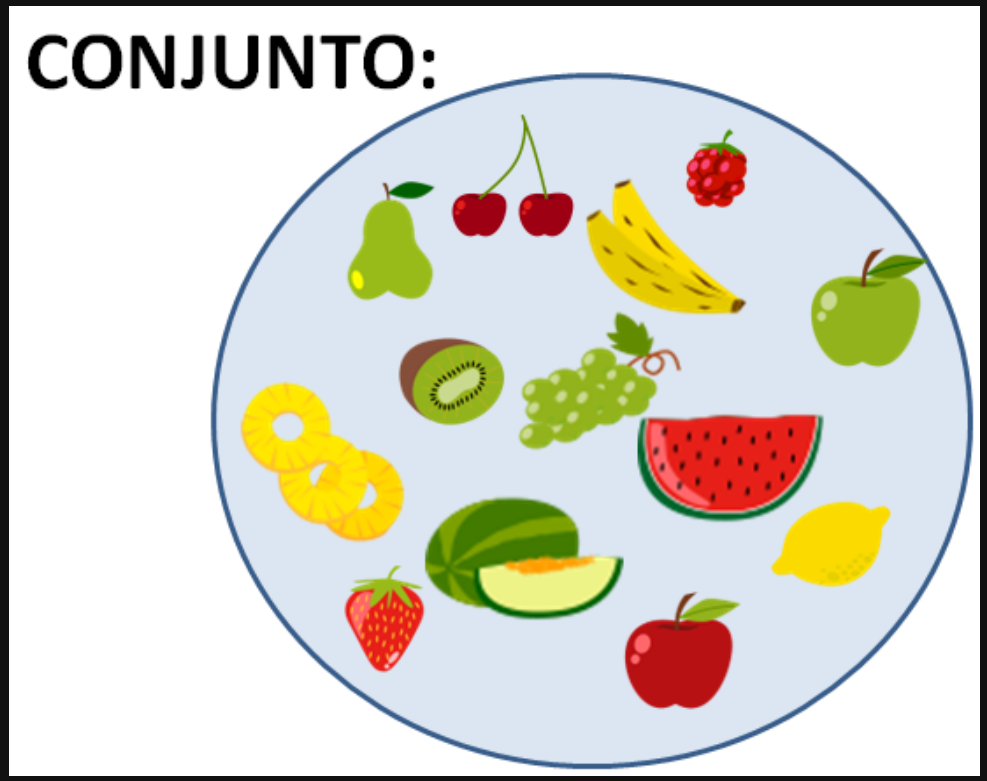
* http://www.cplusplus.com/reference/queue/priority_queue/

>>> queue - principais métodos

* size(): número de elementos armazenados
* empty(): se está vazio
* push(): insere um elemento
* pop(): tira um elemento
* top(): retorna o elemento da frente

>>> Conjunto - set

* Conjunto



* cplusplus.com: "Sets are containers that store unique elements following a specific order."

>>> set - Declaração

```cpp
#include <iostream>
#include <set>
using namespace std;

bool comp (int a, int b) {return a>b;}

int main ()
{
  set<int> s1;                                // set vazio
  int myints[]= {10,20,30,40,50};
  set<int> s2 (myints,myints+5);             // elementos de myints
  set<int> s3 (s2);                        // copia de s2
  set<int> s4 (s2.begin(), s2.end());  // copia de s2
  set<int,bool(*)(int,int)> s5 (myints,myints+5,comp);  // funcao de comparacao

  for(auto e:s2) cout << ' ' << e; cout << endl; //10 20 30 40 50
  for(auto e:s5) cout << ' ' << e; cout << endl; //50 40 30 20 10
  return 0;
}
```

>>> set - Inserção, remoção, acesso e verificação

```cpp
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  set<int> s;
  s.insert (100); //{100}
  s.insert (200); //{100,200}
  s.insert (300); //{100,200,300}
  s.insert (400); //{100,200,300,400}
  s.insert (400); //{100,200,300,400}
  s.erase (s.begin()); //{200,300,400}
  s.erase(400); //{200,300}
  s.size(); //2
  s.clear(); //{}
  return 0;
}
```

>>> set - Buscas

```cpp
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  set<int> s;
  s.insert (100); s.insert (200); s.insert (300); s.insert (400);
  s.count(0); //0
  s.count(100); //1
  s.lower_bound (200); //iterador para 200 (maior elemento <=)
  s.upper_bound (300); //iterador para 400 (menor elemento >)
  s.equal_range(300); //pair <s.lower_bound(200),s.upper_bound(200)>
  return 0;
}
```

* http://www.cplusplus.com/reference/set/set/

>>> set - principais métodos

* begin() e end(): iteradores para o começo e fim
* size(): número de elementos armazenados
* empty(): se está vazio
* find(): encontra um elemento
* lower_bound(): maior elemento menor ou igual a um valor
* upper_bound(): menor elemento maior que um valor
* insert(): insere elementos
* erase(): apaga elementos
* clear(): remove todos os elementos

>>> Exercício

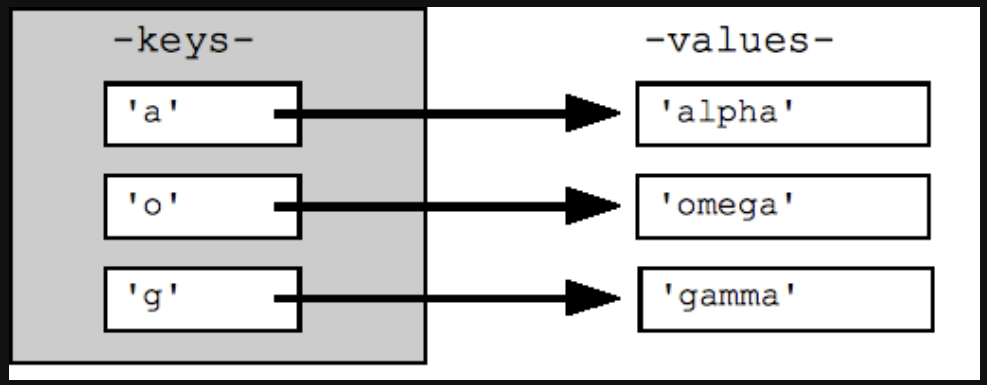
Faça um programa que receba número do usuário (até o fim do arquivo, EOF) e ao final imprima se ele digitou somente números únicos ou se teve algum repetido.

>>> Exercício

```cpp
using namespace std;
int main() {
    int n=0, num;
    set<int> s;
    while(cin >> num) {
        s.insert(num); n++;
    }
    if(n > s.size()) cout << "REPETIDOS" << endl;
    else cout << "ÚNICOS" << endl;
    return 0;
}
```

>>> Dicionário - map

* Dicionário



* cplusplus.com: "Maps (...) store (...) key value and a mapped value, following a specific order."

>>> map - Declaração

```cpp
#include <iostream>
#include <map>
using namespace std;

int main ()
{
  map<char,int> m1;

  m1['a']=10;
  m1['b']=30;
  m1['c']=50;
  m1['d']=70;

  map<char,int> m2 (m1.begin(),m1.end()); // copia m1
  map<char,int> m3 (m2); // copia m2

  for(auto e:m3)  //a:10, b:30, c:50, d:70,
    cout << ' ' << e.first << ':' << e.second << ',';

  cout << endl << m3.count('a');//1

  return 0;
}
```

* http://www.cplusplus.com/reference/map/map/

>>> Exercício

Crie uma função map<char, int> fequencia(const string& texto) que recebe uma string texto. A função deve contar a **frequência** de cada **letra** e retornar um **map** onde a chave é a letra e o valor é o número de vezes que essa letra aparece. Considere que a contagem deve ser feita de forma case-insensitive.

>>> Exercício

```cpp
#include <map>
#include <string>
using namespace std;
map<char, int> frequencia(const string& texto) {
    map<char, int> contagem;
    for(char c : texto) contagem[c]++;
    return contagem;
}
```

>>> Ponteiro

Ponteiros: coisas chatas e perigosas... Déjà Vu?

>>> Ponteiros... de novo...

* Ponteiros nos lembra de...
    * Alocação dinâmica
    * Liberação de memória
    * Vazamento de memória
* Isso é difícil, mas pode ser impossível
    * Exceções

>>> Ponteiros auto gerenciados

* Desalocação automática ao final do escopo
* unique_ptr: não podem existir dois deles apontando para um mesmo endereço
* Definido apenas construtor de movimento, não de cópia

```
void outro(vector<Figura *> &figs) {
  unique_ptr<Figura> nr(new Retangulo(4, 5));

  nr->desenha(); // Sem problemas, mesmo que desenha jogue exce
  // se desenha() joga exceção, o novo retangulo é apagado

  figs.push_back(nr.release()); // aqui o controle é entregue a
}
```

>>> Ponteiros auto gerenciados

* **shared_ptr**: mais de um **podem apontar para o mesmo endereço**

```cpp
#include <iostream>
#include <memory>

class Example {
public:
    Example() {
        std::cout << "Objeto Example criado" << std::endl;
    }
    ~Example() {
        std::cout << "Objeto Example destruído" << std::endl;
    }
    void showMessage() const {
        std::cout << "Mensagem do objeto Example" << std::endl;
    }
};
```

>>> Ponteiros auto gerenciados

```cpp
int main() {
    // Cria o objeto
    std::shared_ptr<Example> ptr1 = std::make_shared<Example>();
    {
        std::shared_ptr<Example> ptr2 = ptr1;
        ptr2->showMessage();
        std::cout << "Contagem de referência: "
                  << ptr1.use_count() << std::endl;
    }

    std::cout << "Contagem de referência após o escopo: "
              << ptr1.use_count() << std::endl;
    ptr1->showMessage();
    return 0;
}
```

* Aulas do grupo Maratona IFSC (Eu e Ian Giesta)
* A very modest STL tutorial
* cplusplus.com