



# Introdução a Técnicas de Programação

## Funções

Prof. André Campos  
DIMAp/UFRN

# Clínica de saúde

Durante as 4 horas que você esperou para ser atendido por um médico numa clínica de saúde, você percebeu que, além de você, muitos iam na recepcionista perguntar quantos ainda faltam antes de ser atendido, principalmente porque algumas vezes pessoas passavam na sua frente. O fato é que sintomas mais graves têm prioridade e devem ser atendidos logo. Porém, sem essa informação, muitos como você ficam chateados e perdem a paciência. Seria muito mais conveniente se todos os pacientes da clínica pudessem consultar essa informação no celular.

Você viu que tinha uma oportunidade de empreendimento e resolveu fazer um sistema de acompanhamento. Para isso, é necessário representar uma fila em que pessoas podem “entrar na frente”.





## Possível solução

1. Defina duas filas (uma prioritária e outra não)
2. Leia o número de atendimentos
3. Enquanto houver atendimento a realizar
  - a. Leia o código de chegada
  - b. Se for um novo paciente prioritário, insira no final da fila prioritária
  - c. Se for um novo paciente sem prioridade, insira no final não-prioritária
  - d. Se for um atendimento
    - i. Se tiver gente na fila prioritária, remova o primeira da prioritária
    - ii. Senão remova o primeira da não-prioritária
    - iii. Aumente 1 no número de atendimentos ocorridos
4. Imprima os pacientes da fila prioritária
5. Imprima os pacientes da fila normal



## Possível solução

1. Defina duas filas (uma prioritária e outra não)
2. Leia o número de atendimentos
3. Enquanto houver atendimento a realizar
  - a. Leia o código de chegada
  - b. Se for um novo paciente prioritário, **insira no final da fila prioritária**
  - c. Se for um novo paciente sem prioridade, **insira no final não-prioritária**
  - d. Se for um atendimento
    - i. Se tiver gente na fila prioritária, **remova o primeira da prioritária**
    - ii. Senão **remova o primeira da não-prioritária**
    - iii. Aumente 1 no número de atendimentos ocorridos
4. **Imprima os pacientes da fila** prioritária
5. **Imprima os pacientes da fila** normal

# Sub-rotinas

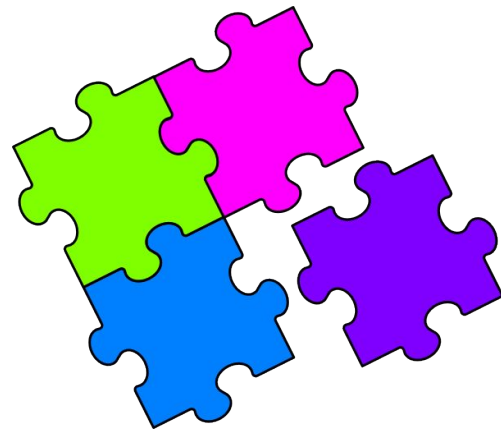
Em C++, temos a **rotina principal** (`main()`). A partir dela, podemos chamar outras rotinas secundárias... ou **sub-rotinas**.

Sub-rotina é uma sequência de instruções organizadas para **facilitar o reuso**

Ajuda a pensar sobre o **problema maior a partir de pedaços menores**

Mini-programas que resolvem problemas específicos

**Facilita testar** trechos de códigos (funcionalidades)



# Definindo nossa própria sub-rotina

Modelo

## Elementos

- Nome (identificador único)
- Conjunto de parâmetros (opcional)
  - com o tipo e nome de cada um
- Tipo do dado que será retornado
- Sequência de instruções (corpo da sub-rotina)
- Retorno do valor com a instrução **return**

```
tipo de retorno nome(parâmetros) {  
    instruções  
    ...  
    return valor;  
}
```

Exemplo

```
int maior(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    return b;  
}
```



# Funções e procedimentos

Em outras linguagens, uma sub-rotina que

- não retorna nada, pode ser chamada de “procedimento”
- retorna algo, pode ser chamada de “função”

Em C++, tudo é função!

Mas quando não queremos retornar algo, especificamos o tipo de retorno como **void** (vazio ou nulo)

Podemos usar **return** em qualquer parte do sub-rotina

Não necessariamente no final



# Exemplo de função sem retorno (procedimento)

O tipo de retorno é **void**

Usa-se o **return** sem valor algum, se desejar retornar de um ponto do código.

```
/* Imprime um dado caractere várias vezes. */  
  
void printMany(char c, int n) {  
    for (int i = 0; i < n; i++) {  
        cout << c;  
    }  
    return; // nem precisa porque termina aqui  
}
```

```
/* Exemplo de uso. */  
  
int main() {  
    printMany('#', 5);  
    return 0;  
}
```





# Etapas na construção de uma função

Escreva uma função que recebe 3 valores inteiros e retorna o maior deles.



# Etapas na construção de uma função

parâmetros

valor de retorno

Escreva uma função que recebe 3 valores inteiros e retorna o maior deles.

- 1) Identifique os parâmetros (3 int) e o tipo de retorno (int)



# Etapas na construção de uma função

Escreva uma função que recebe **3 valores inteiros** e **retorna o maior** deles.

- 1) Identifique os parâmetros (3 int) e o tipo de retorno (int)
- 2) Escolha um nome (identificador) para a função e defina casos para saber como ela deve funcionar

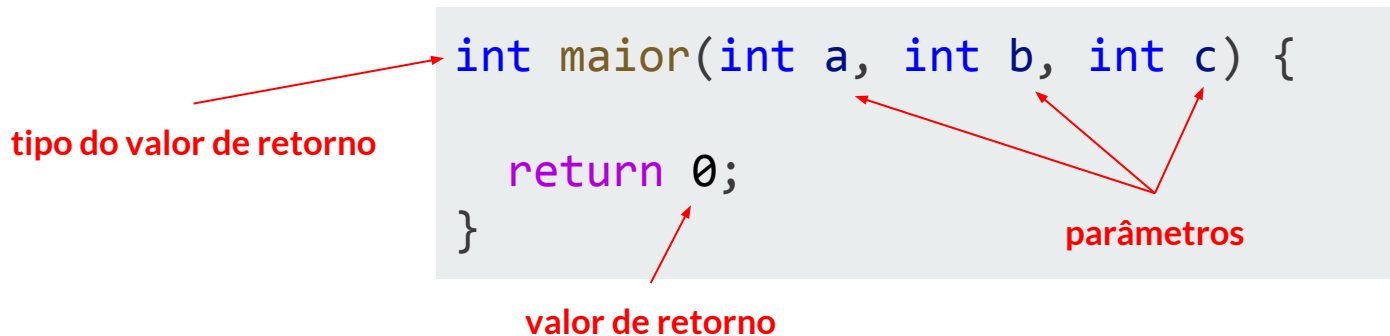
## Testes

Chamada	Valor de retorno
<code>maior(5, 2, 7)</code>	7
<code>maior(1, 8, 4)</code>	8
<code>maior(-7, -4, -1)</code>	-1

# Etapas na construção de uma função

Escreva uma função que recebe 3 valores inteiros e retorna o maior deles.

- 1) Identifique os parâmetros (3 int) e o tipo de retorno (int)
- 2) Escolha um nome (identificador) para a função e defina casos para saber como ela deve funcionar
- 3) Escreva a estrutura da função, ainda sem sua implementação (sua lógica)



The diagram shows a C function definition: `int maior(int a, int b, int c) {  
 return 0;  
}`. Red arrows point from text labels to parts of the code: 'tipo do valor de retorno' points to 'int' at the start; 'parâmetros' points to 'a, b, c'; 'valor de retorno' points to '0'.

```
int maior(int a, int b, int c) {  
    return 0;  
}
```

tipo do valor de retorno

parâmetros

valor de retorno

COMPILE !

# Etapas na construção de uma função

Escreva uma função que recebe 3 valores inteiros e retorna o maior deles.

- 4) Escreva a lógica da função, atentando que todos os valores de retorno devem atender o tipo de retorno!

```
int maior(int a, int b, int c) {  
    if (a > b and a > c) return a;  
    if (b > c) return b;  
    return c;  
}
```

tipo do valor de retorno

valores de retorno

COMPILE !



## Etapas na construção de uma função

Escreva uma função que recebe **3 valores inteiros** e **retorna o maior** deles.

- 4) Escreva a lógica da função, atentando que todos os valores de retorno devem **atender o tipo de retorno!**
- 5) Crie chamadas para função com os valores dos testes

```
int main() {  
    cout << maior(5, 2, 7) << endl;  
    cout << maior(1, 8, 4) << endl;  
    cout << maior(-7, -4, -1) << endl;  
    return 0;  
}
```

```
int maior(int a, int b, int c) {  
    if (a > b and a > c) return a;  
    if (b > c) return b;  
    return c;  
}
```

# Etapas na construção de uma função

Escreva uma função que recebe **3 valores inteiros** e **retorna o maior** deles.

- 4) Escreva a lógica da função, atentando que todos os valores de retorno devem **atender o tipo de retorno!**
- 5) Crie chamadas para função com os valores dos testes
- 6) Verifique se as saídas dos testes correspondem ao esperado

```
int main() {  
    cout << maior(5, 2, 7) << endl;  
    cout << maior(1, 8, 4) << endl;  
    cout << maior(-7, -4, -1) << endl;  
    return 0;  
}
```

Chamada	Valor de retorno
maior(5, 2, 7)	7
maior(1, 8, 4)	8
maior(-7, -4, -1)	-1



# Exemplo

Escreva uma função que recebe uma string e retorna uma nova string com a string passada invertida.

- 1) Identifique os parâmetros (1 string) e o tipo de retorno (string)
- 2) Escolha um nome (identificador) para a função e defina casos para saber como ela deve funcionar

## Testes

Chamada	Valor de retorno
<code>inverte("abcde")</code>	<code>"edcba"</code>
<code>inverte("aa b cc")</code>	<code>"cc b aa"</code>
<code>inverte("")</code>	<code>""</code>





## Exemplo

Escreva uma função que recebe uma string e retorna uma nova string com a string passada invertida.

- 1) Identifique os parâmetros (1 string) e o tipo de retorno (string)
- 2) Escolha um nome (identificador) para a função e defina casos para saber como ela deve funcionar
- 3) Escreva a estrutura da função, ainda sem sua implementação (sua lógica)

```
string inverta(string s) {  
    return s;  
}
```

COMPILE !

# Exemplo

Escreva uma função que recebe uma string e retorna uma nova string com a string passada invertida.

- 4) Escreva a lógica da função, atentando para **o tipo de retorno**
- 5) Crie chamadas para função com os valores dos testes

COMPILE !

```
int main() {  
    cout << inverte("abcde") << endl;  
    cout << inverte("aa b cc") << endl;  
    cout << inverte("") << endl;  
    return 0;  
}
```

```
string inverte(string s) {  
    string r = "";  
    for(char c: s) {  
        r = c + r;  
    }  
    return r;  
}
```

# Exemplo

Escreva uma função que recebe uma string e retorna uma nova string com a string passada invertida.

- 4) Escreva a lógica da função, atentando para **o tipo de retorno**
- 5) Crie chamadas para função com os valores dos testes
- 6) Verifique se as saídas dos testes correspondem ao esperado

```
int main() {  
    cout << inverte("abcde") << endl;  
    cout << inverte("aa b cc") << endl;  
    cout << inverte("") << endl;  
    return 0;  
}
```

Chamada	Valor de retorno
inverte("abcde")	"edcba"
inverte("aa b cc")	"cc b aa"
inverte("")	" "

# Escopo de variáveis

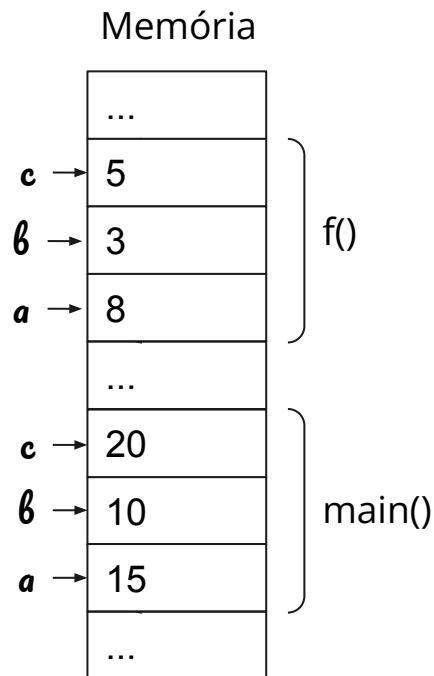
As variáveis definidas dentro de uma sub-rotina existem exclusivamente para aquela sub-rotina.

Uma área da memória é reservada quando uma função é chamada.

Suas variáveis residem lá e são apagadas quando a função termina.

```
void f() {  
    int a = 8;  
    int b = 3;  
    int c = 5;  
}
```

```
int main() {  
    int a = 15;  
    int b = 10;  
    int c = 20;  
    f();  
}
```



# Escopo local vs escopo global

## Variáveis de escopo local (ou variáveis locais)

Variáveis definidas em uma função são visíveis apenas dentro da função.

## Variáveis de escopo global (ou variáveis globais)

São definidas fora das funções e podem ser utilizadas por qualquer função.

*Obs 1: Se uma variável local tiver o mesmo nome de uma global, será dada preferência para a local.*

*Obs 2: Quanto menos variáveis globais, mais organizado estará seu código (princ. de isolamento). **Evite usar!***

```
int a = 8; // global
int b = 8; // global

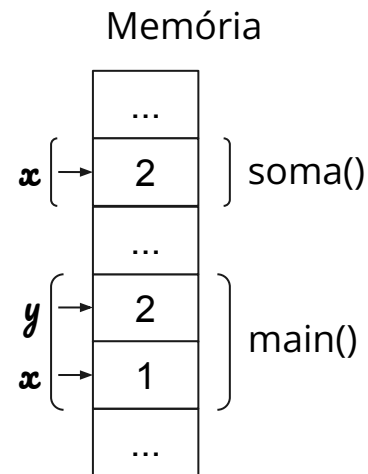
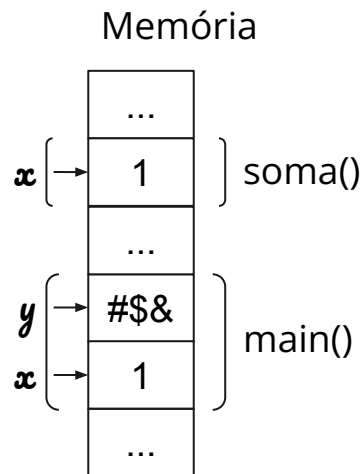
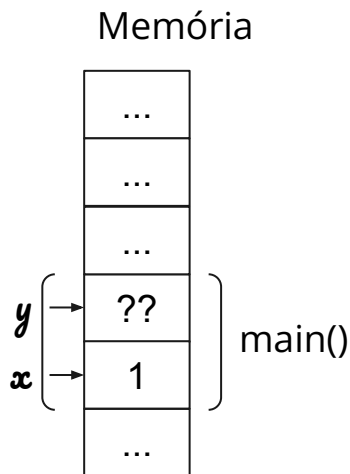
void f() {
    cout << a << " " << b;
}

int main() {
    int a = 15; // local
    cout << a << " " << b;
    f();
    return 0;
}
```

# Passagem de parâmetros por valor

Os parâmetros são variáveis **locais**, que recebem os valores do que é passado na chamada (**são cópias**).  
As modificações realizadas na função **não alteram** as variáveis passadas, mesmo que tenham **o mesmo nome**.

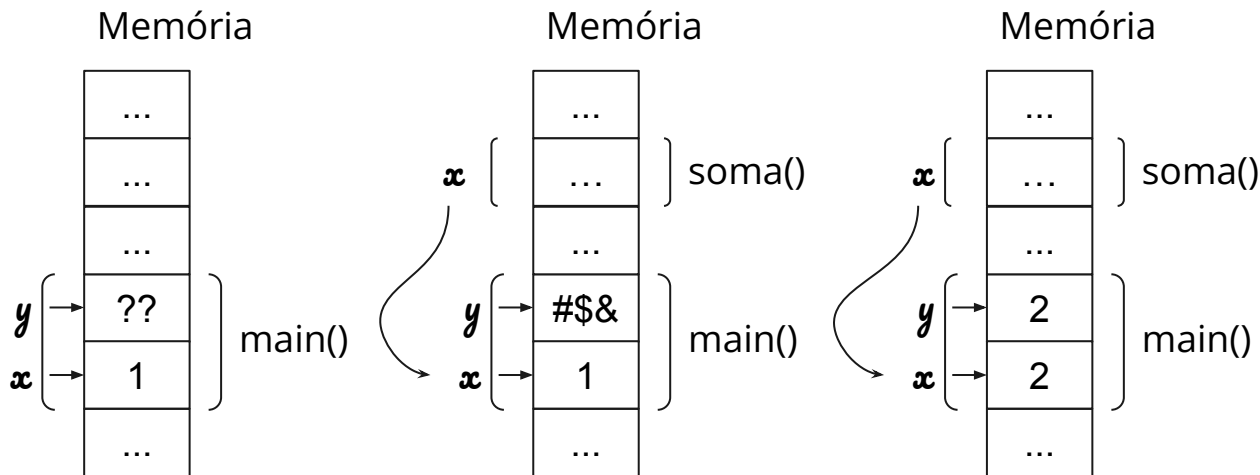
```
int soma(int x) {  
    x++;  
    cout << x << endl;  
    return x;  
}  
  
int main() {  
    int x = 1;  
    int y = soma(x);  
    cout << x << endl;  
    cout << y << endl;  
}
```



# Passagem de parâmetros por referência

Os parâmetros são variáveis **que fazem referência** às variáveis passadas na chamada (**não são cópias**).  
As modificações realizadas na função **alteram** as variáveis passadas, mesmo que tenham o **nome diferente**.

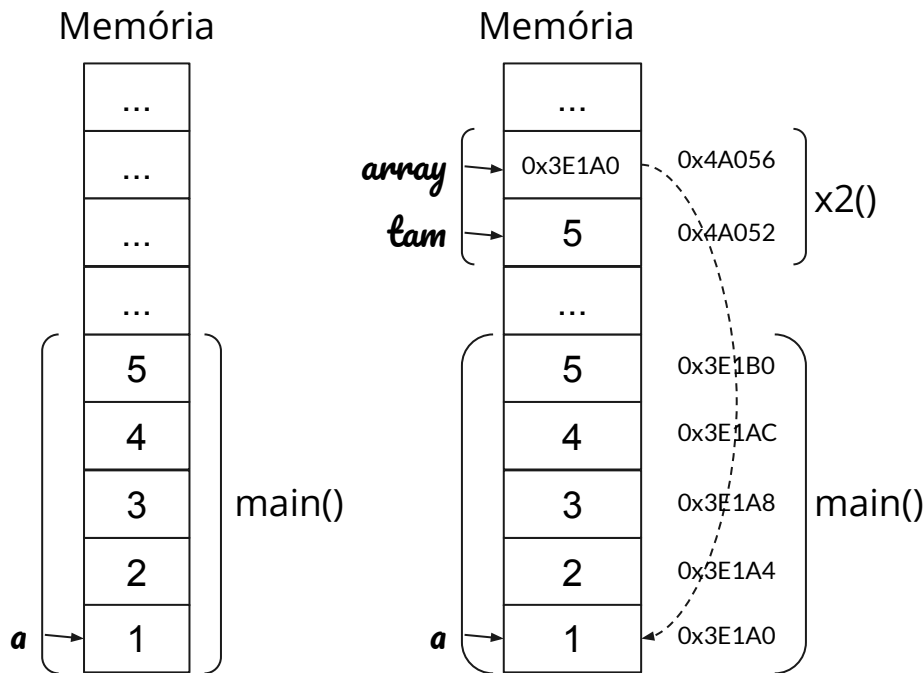
```
int soma(int &x) {  
    x++;  
    cout << x << endl;  
    return x;  
}  
  
int main() {  
    int x = 1;  
    int y = soma(x);  
    cout << x << endl;  
    cout << y << endl;  
}
```



# Passagem de arrays

As variáveis arrays são endereços de memória. Assim, quando passamos a variável, passamos o endereço onde ela está armazenada. Logo, alterações nos valores nos índices refletem nas variáveis originais.

```
void x2(int tam, int array[]) {  
    for (int i = 0; i < tam; i++) {  
        array[i] *= 2;  
    }  
}  
  
int main() {  
    int a[] = { 1, 2, 3, 4, 5 };  
    x2(5, a);  
    return 0;  
}
```







## Exemplo de função com passagem de array

Escreva uma função para buscar um valor inteiro em um array de inteiros. Se o valor se encontrar no array, a função deve retornar o índice que ele se encontra, caso contrário deve retornar o valor -1 (não há índice com esse valor).



## Exemplo de função com passagem de array

Escreva uma função para buscar um valor inteiro em um array de inteiros. Se o valor se encontrar no array, a função deve retornar o índice que ele se encontra, caso contrário deve retornar o valor -1 (não há índice com esse valor).

1) Parâmetros: **int valor**, **int tamanho\_array**, **int array[]**

Retorno: **int** (índice da posição ou -1)

2)

Chamada	Valor de retorno
busca (5, 4, { 2, 4, 5, 1 })	2
busca (3, 2, { 2, 4 })	-1
busca (5, 0, { })	-1

## Exemplo de função com passagem de array

Escreva uma função para buscar um valor inteiro em um array de inteiros. Se o valor se encontrar no array, a função deve retornar o índice que ele se encontra, caso contrário deve retornar o valor -1 (não há índice com esse valor).

3)

```
int busca(int val, int n, int v[]) {  
    return -1;  
}
```

COMPILE !

4)

```
int busca(int val, int n, int v[]) {  
    for (int i = 0; i < n; i++) {  
        if (val == v[i]) return i;  
    }  
    return -1;  
}
```

COMPILE !

## Exemplo de função com passagem de array

Escreva uma função para buscar um valor inteiro em um array de inteiros. Se o valor se encontrar no array, a função deve retornar o índice que ele se encontra, caso contrário deve retornar o valor -1 (não há índice com esse valor).

5)

```
int main() {  
    int a1[] = { 2, 4, 5, 1 };  
    int a2[] = { 2, 4 };  
    int a3[] = {};  
    cout << busca(5, 4, a1) << endl;  
    cout << busca(3, 2, a2) << endl;  
    cout << busca(5, 0, a3) << endl;  
    return 0;  
}
```

6)

Chamada	Retorno
busca(5, 4, {2, 4, 5, 1})	2
busca(3, 2, {2, 4})	-1
busca(5, 0, {})	-1



# ATENÇÃO! Retorno de arrays

**NÃO RETORNEM ARRAYS DEFINIDOS LOCALMENTE**, pois quando a função termina, as variáveis locais são liberadas.

Mais detalhes serão apresentados quando falarmos de ponteiros e memória stack.

Por enquanto, se for necessário “retornar” um array, passe ele por parâmetro e altere os valores nos seus índices.

```
void prog_aritmetica(int inicio, int razao, int n, int a[]) {  
    for(int i = 0; i < n; i++) {  
        a[i] = inicio + i * razao;;  
    }  
}
```



# Valores padrões nos parâmetros

Permite que um parâmetro assuma um valor, caso não seja passado nenhum na sua chamada.

Os parâmetros que possuem valores padrões devem ser os últimos parâmetros da função.

```
int max(int a, int b = 0) {  
    return a > b ? a : b;  
}  
  
int main() {  
    cout << max(3,5);  
    cout << max(4);  
    cout << max(-5);  
    return 0;  
}
```

## Voltando ao problema

1. Defina duas filas (uma prioritária e outra não)
2. Leia o número de atendimentos
3. Enquanto houver atendimento a realizar
  - a. Leia o código de chegada
  - b. Se for um novo paciente prioritário, insira no final da fila prioritária
  - c. Se for um novo paciente sem prioridade, insira no final não-prioritária
  - d. Se for um atendimento
    - i. Se tiver gente na fila prioritária, remova o primeira da prioritária
    - ii. Senão remova o primeira da não-prioritária
    - iii. Aumente 1 no número de atendimentos ocorridos
4. Imprima os pacientes da fila prioritária
5. Imprima os pacientes da fila normal





# Funções da fila

```
/* insere 'value' no final do array 'queue' */
void push(int value, int queue[], int &size) {
    queue[size] = value;
    size++;
}

/* retira e retorna o 1º valor de 'queue' */
int pop(int queue[], int &size) {
    int top = queue[0];
    for(int i = 1; i < size; i++) {
        queue[i-1] = queue[i];
    }
    size--;
    return top;
}
```

```
/* imprime os valores em 'queue' */
void print(int queue[], int size) {
    for(int i = 0; i < size; i++) {
        cout << queue[i] << " ";
    }
    cout << endl;
}
```





# Uso das funções da fila

```
int main() {
    int non_prior[MAX], prior[MAX], size_prior = 0, size_non_prior = 0, code, num;
    cin >> num;
    while (num > 0) {
        cin >> code;
        if (code == 0) {
            if (size_prior > 0) pop(prior, size_prior);
            else pop(non_prior, size_non_prior);
            num--;
        }
        else if (code < 100) push(code, non_prior, size_non_prior);
        else push(code, prior, size_prior);
    }
    print(prior, size_prior);
    print(non_prior, size_non_prior);
    return 0;
}
```

# “Assinatura” (ou interface) de uma função

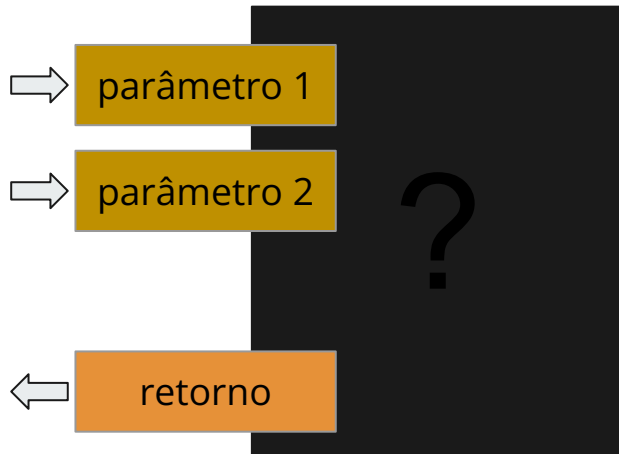
Funções encapsulam o processamento dos dados (caixa-preta)

Para usá-las, basta saber “o que passar” e “o que receber”

É o que chamamos de **assinatura** ou **interface**.

*É necessário para a compilação de um código que usa a função*

```
/* insere 'value' no final do array 'queue' */  
void push(int value, int queue[], int &size);  
  
/* retira e retorna o 1º valor de 'queue' */  
int pop(int queue[], int &size);  
  
/* imprime os valores em 'queue' */  
int print(int queue[], int size);
```





# Práticas

1. Escreva uma função que recebe dois parâmetros, um float B e outro inteiro não negativo P e retorna a potência  $B^P$ .

```
float power(float, int);
```

2. Escreva uma função que recebe um vetor de inteiros e o número de elementos dele e retornar o maior valor do vetor.

```
int max(int[], int);
```

3. Escreva uma função que recebe dois valores inteiros e retorna seu MDC.

```
int mdc(int, int);
```



## Práticas

4. Escreva uma função que recebe dois conjuntos de tamanho N e M, representados por vetores de números inteiros, e retorna verdadeiro se há interseção entre os conjuntos ou falso se são conjuntos disjuntos.

```
bool hasIntersection(int n, int a[], int m, int b[]);
```

5. Escreva uma função que calcula a distância entre dois pontos 2D. Será necessário usar a função matemática `sqrt()` para a raiz quadrada (use `#include <cmath>`)

```
float distance(float x1, float y1, float x2, float y2);
```

6. Escreva uma função que recebe uma sequência de pontos, representados por dois vetores, um contendo as coordenadas X e o outro as Y, e retorna a menor distância entre os pontos. Reutilize a solução da atividade 5.

```
float minDist(int num, float x[], float y[]) ;
```