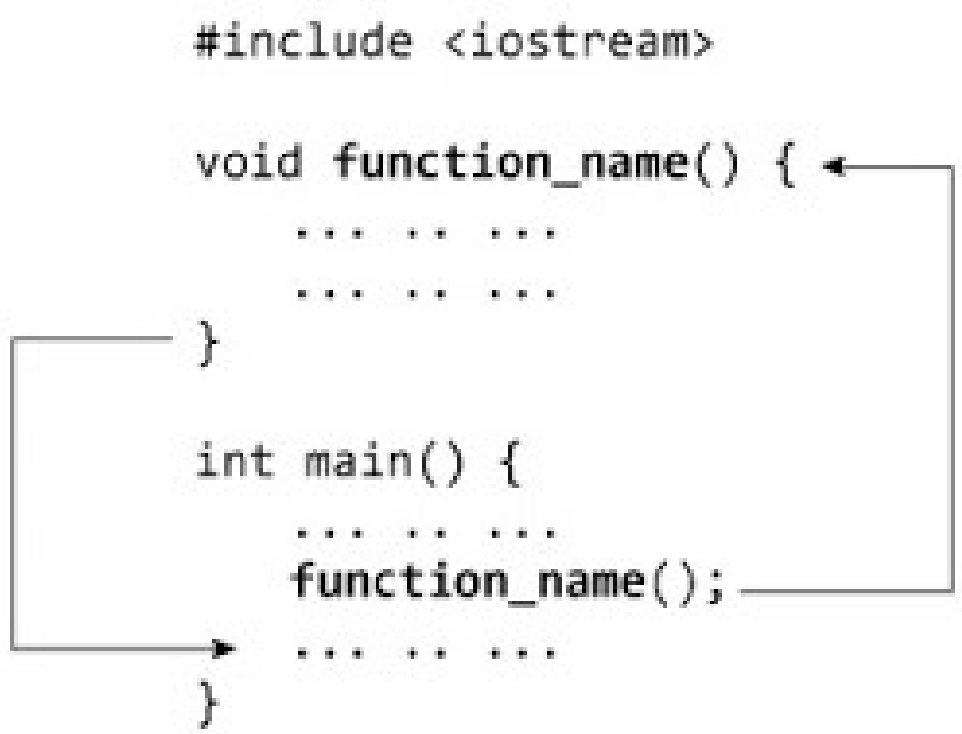


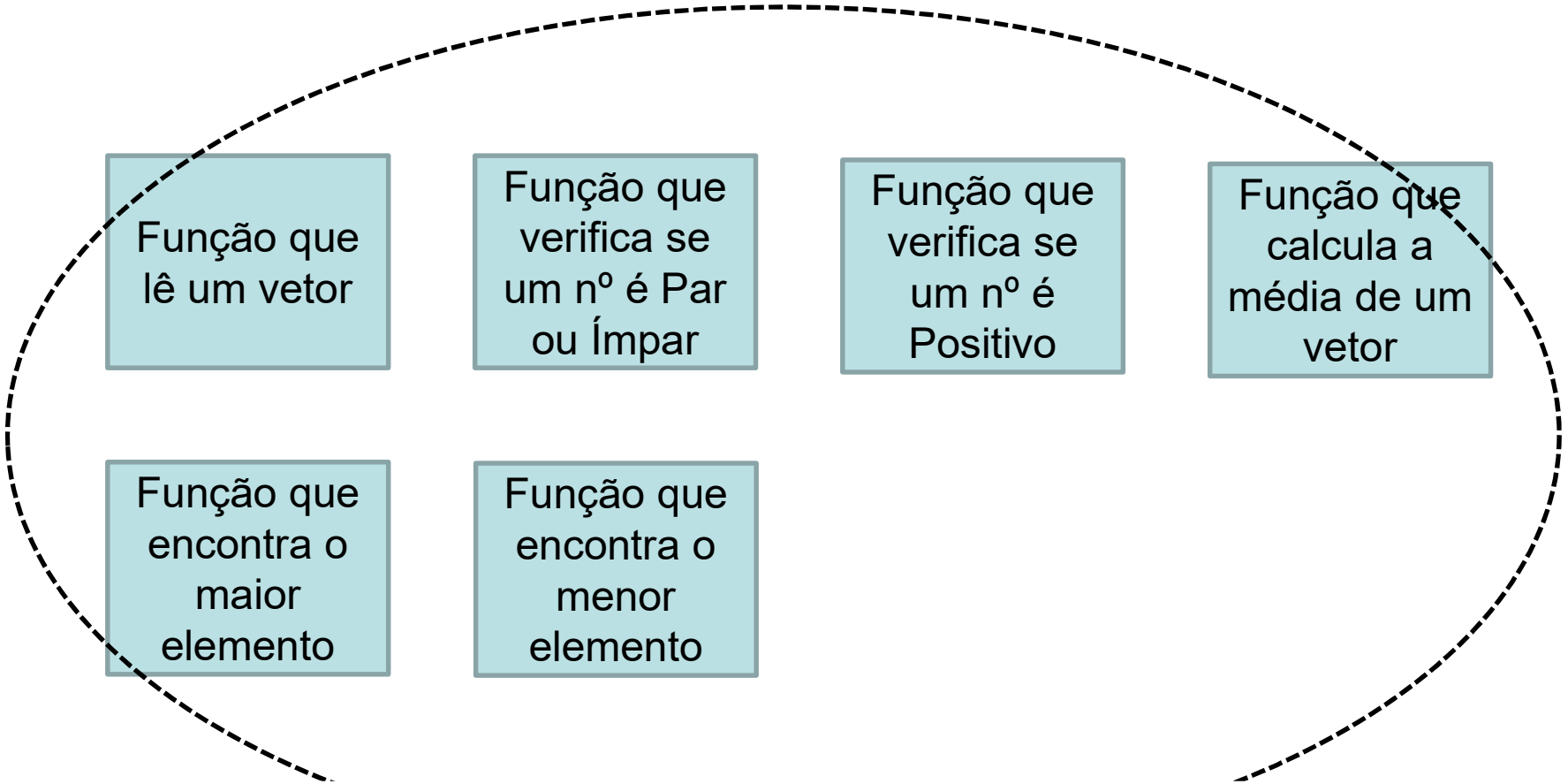
1º ADS – AED I

FUNÇÃO

Funções – Introdução

- Importante recurso apresentado nas linguagens de programação:
  - Modularização: um programa pode ser particionado em sub-rotinas específicas.
- Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa bem definida.
- C++ possui a função main(), por onde começa a execução do programa.

Funções – Introdução



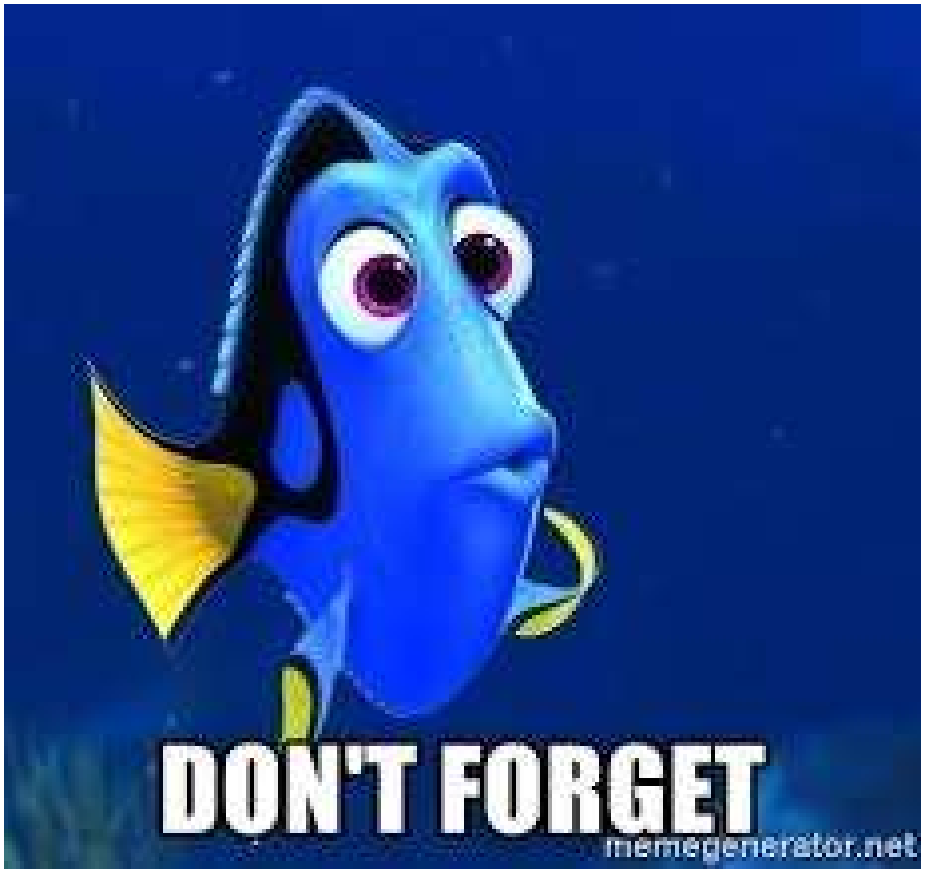
Funções – Introdução (cont)

- Existem outras funções predefinidas na linguagem C++:
  - clrscr()
  - strcpy()
  - pow(n1, n2)
  - sqrt(num)
  - getch()

Função – definição

- Função é um conjunto de comandos que executam determinadas tarefas;
- Uma vez definida, uma função poderá ser chamada várias vezes em um programa e em pontos diferentes, evitando a repetição de código-fonte e implicando em sua modularização.

Toda função deve retornar um valor.



Toda função deve retornar um valor!

Função – definição

- O que acontece quando executamos a função sqrt()?

```
…
x = sqrt(9);
cout << “A raiz de 9 e´: “ << x;
…
```

C++ executa a função sqrt() que está na biblioteca <math.h> e retorna a raiz quadrada de 9.

Nós não vemos o código fonte da função sqrt(). Nós **chamamos** a função, ela é **executada** e **retorna** o resultado.

O resultado (retorno) da função é armazenado na variável x.

Forma Geral

tipo nome_da_função (**argumentos**)

{

    Instr 1;

    Instr 2;

    Instr n;

}

Onde:

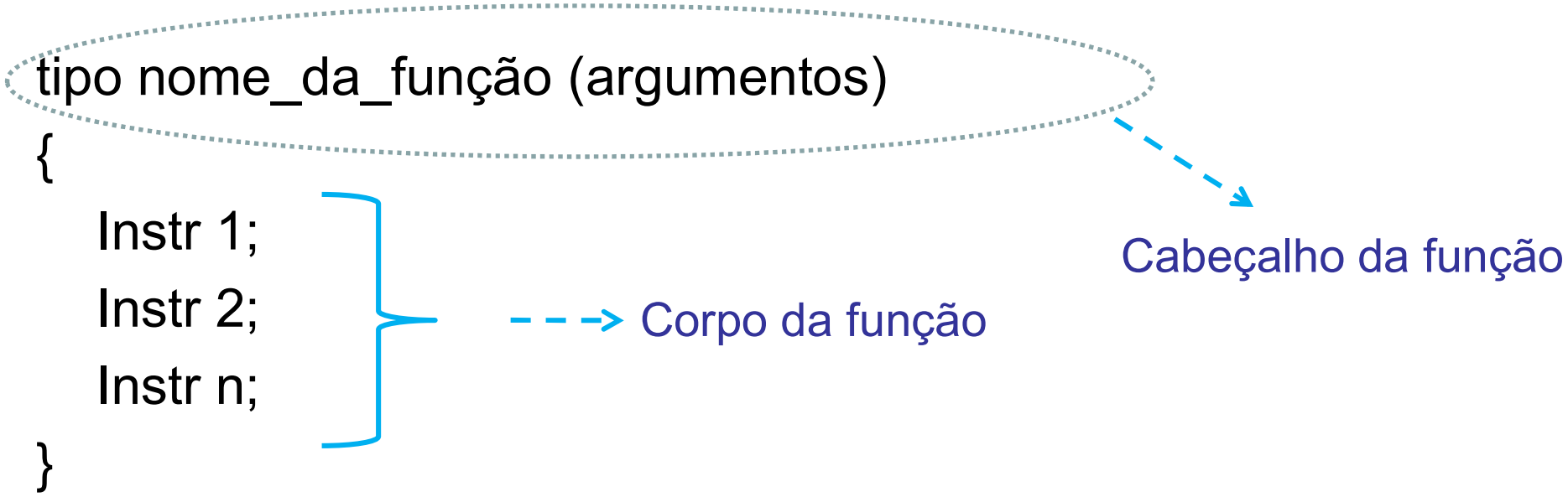
tipo ⇨ tipo do valor a ser retornado pela função.

Argumentos ⇨ argumentos passados à função, separados por vírgula.

{ ⇨ início do corpo da função.

} ⇨ fim do corpo da função.

Forma Geral (cont)



Tipos Primitivos

- Int
- Float
- Char
- Double
- Bool
- **Void**

Tipos Primitivos

- **Void**
  - O tipo void é o tipo para o resultado de uma função que retorna normalmente, mas não fornece um valor de resultado para quem a chamou.
  - O tipo void quer dizer vazio. Ele nos permite escrever funções que não retornam nada e funções que não têm parâmetros.

Retorno de valor e recepção de argumentos

I – Funções que não retornam valor e nem recebem argumentos

II – Funções que recebem argumentos e não retornam valor

III – Funções que recebem argumentos e retornam valor

IV – Funções que retornam valor sem receber argumentos

Visibilidade de variáveis

- Variável local ou privada
- Variável global ou pública

Visibilidade de variáveis

- Variável **local** ou **privada**
  - São aquelas que só têm validade dentro do bloco (função) no qual são declaradas.
  - Podemos declarar variáveis dentro de qualquer bloco (função).
  - Variáveis locais não são reconhecidas fora da função onde foram definidas.
  - A declaração de variáveis locais é a primeira coisa que devemos colocar numa função.
  - A característica que torna as variáveis locais tão importantes é justamente a de serem exclusivas da função.

```
func1 (...) {
        int abc,x;
        ...
}

func2 (...) {
        int abc;
        ...
}

main () {
        int a,x,y;
        ...
        ...
}
```

- As variáveis locais de cada função não irão interferir com as variáveis locais de outras funções.
- Assim, a variável abc de func1() não tem nada a ver (e pode ser tratada independentemente) com a variável abc de func2().
- A variável x de func1() é também completamente independente da variável **x** de main().

Visibilidade de variáveis

- Variável **global** ou **pública**
  - Variáveis globais são declaradas fora de todas as funções do programa.
  - Elas são conhecidas e podem ser alteradas por todas as funções do programa.
  - Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.
  - Um exemplo:

```
int z,k;
func1 (...) {
        int x,y;
        ...
}

func2 (...) {
        int x,y,z;
        ...
        z=10;
        ...
}

main () {
        int count;
...
}
```

- As variáveis z e k são globais.
- Todas as funções: func1(), func2() e main(), conseguem enxergar as variáveis z e k;
- Por este motivo elas são chamadas de **variáveis globais**.

- Observação
  - Veja que func2() tem uma variável local chamada z.
    - Quando temos então, em func2(), o comando z=10 quem recebe o valor de 10 é a variável local, não afetando o valor da variável global z.

I – Funções que não retornam valor e nem recebem argumentos

```
#include<iostream>
#include<conio.h>
using namespace std;
#define t 4
int v[t],i; //Variáveis Globais ou Públicas

void quadrado(){
      cout<<"O quadrado de " << v[i] << ":" << v[i]*v[i];
}
```

```
void leitura(){
  for (i = 0; i < t; i++){
     cout << "\n[" << i+1 << "]: ";
     cin >> v[i];
     quadrado();
  }
}

main(){
     leitura();
}
```

As funções são escritas ANTES do main ().

I – Funções que não retornam valor e nem recebem argumentos

Escreva um programa que leia um vetor de 5 elementos, do tipo real. Calcule e mostre:

a) A média entre os elementos do vetor;
b) A quantidade de elementos positivos;
c) A quantidade de elementos negativos;
d) A quantidade de elementos nulos.

```
#include<iostream>
#include<conio.h>
using namespace std;
#define t 5

int v[t],i; //Variáveis Global ou Pública

void nulos(){
    int n=0; //Variável Local ou Privada
    for(i=0;i<t;i++){
       if(v[i] == 0){
          n++;
       }
    }
    cout << "\nA qtde de NULOS: " << n;
}
```

```
main(){
   leitura();
   media();
   positivos();
   negativos();
   nulos();
}
```

Crie uma função para cada item do enunciado:

- leitura()
- media()
- positivos()
- negativos()
- nulos()

A ordem não importa. O *main()* é quem Define a ordem de execução das funções. Defina todas as variáveis como globais.

RESOLUÇÃO COMPLETA EM C++

```
#include<iostream>
#include<conio.h>
using namespace std;
#define t 5

int v[t],i; //Variáveis Globais

void nulos(){
    int n=0; //Variável Local ou Privada
    for(i=0;i<t;i++){
       if(v[i] == 0){
          n++;
       }
    }
    cout << "\nA qtde de NULOS: " << n;
}// Fim da função nulos() – Retorna para quem chamou
```

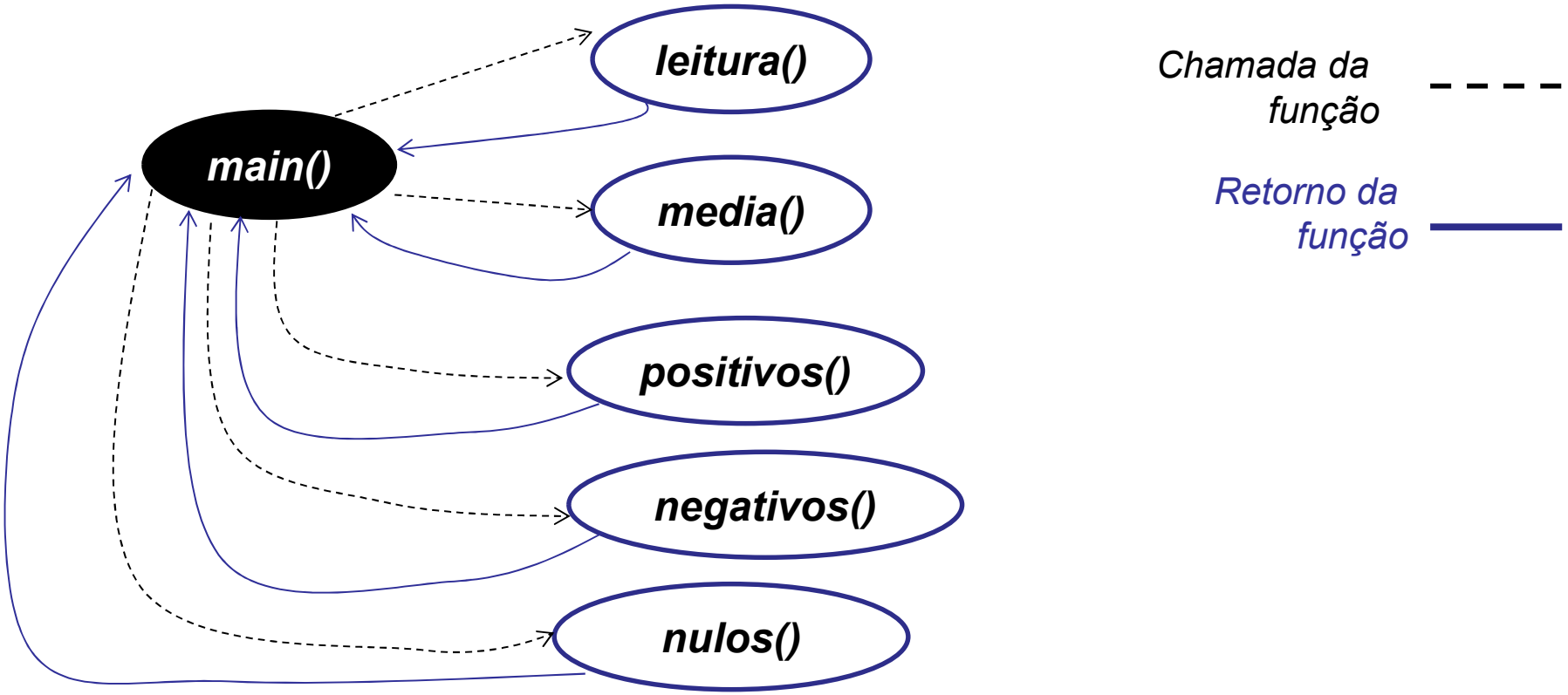
```
void positivos() {
        int p = 0; i = 0;
        while(i < t) {
                if(v[i] > 0) {
                        p++;
                }
                i++;
        }
        cout << "\nA qtde de POSITIVOS: " << p;
} //Fim da função positivos() – Retorna para quem chamou

void leitura() {
        i = 0;
        while(i < t) {
                cout << "Digite o " << i+1 << "o. elemento: ";
                cin >> v[i];
                i++;
        }
} //Fim da função leitura() – Retorna para quem chamou
```

```
void media() {
        float s = 0;
        i = 0;
        while(i < t) {
                s += v[i];
                i++;
        }
        cout << "\nA media: " << s/t;
} //Fim da função media() – Retorna para quem chamou

void negativos() {
   int neg=0; //Variável Local ou Privada
   for(i = 0; i < t; i++) {
      if(v[i] < 0) {
         neg++;
      }
   }
   cout << "\nA qtde de NEGATIVOS: " << neg;
} //Fim da função negativos() – Retorna para quem chamou
```

```
main() { //A execução do programa SEMPRE começa aqui.
   leitura();
   media();
   positivos();
   negativos();
   nulos();
} //Fim da função main() – Fim do programa.
```



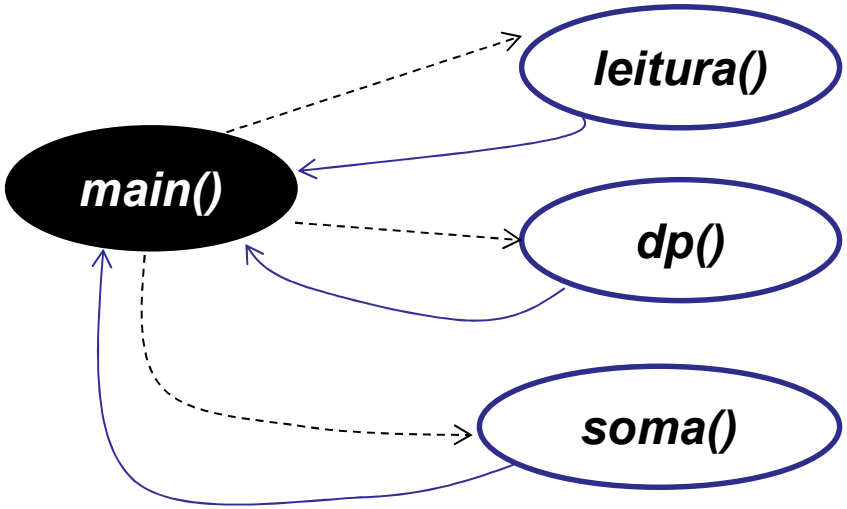
Exercícios

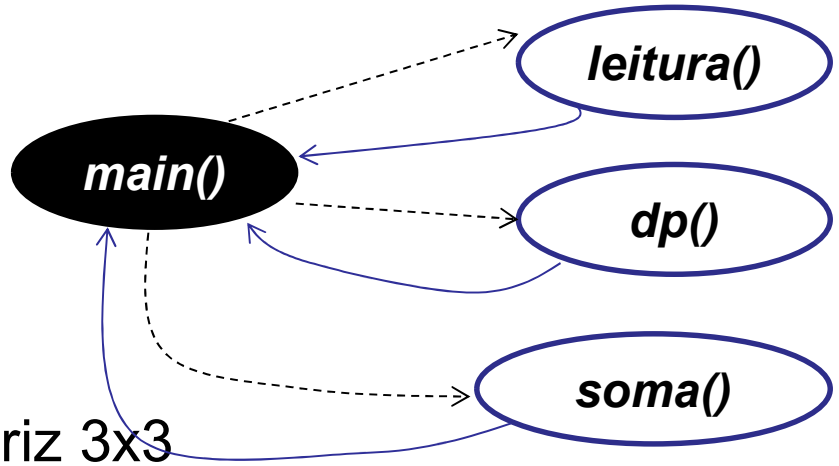
1. Escreva um programa que leia uma matriz de 3x3, do tipo inteiro, e mostre:
   a) Os elementos da diagonal principal (i == j);
   b) A soma entre os elementos que estão abaixo da diagonal principal (i > j).

Utilize recursos de função.

- Neste tipo exercício, a matriz deve ser definida como **global**, mas os índices podem ser definidos como **locais** em cada uma das funções.
- Crie 3 funções: a função leitura(), a função dp() para resolver **(a)** e a função soma() para resolver **(b)**.
- Mostre o resultado de **(a)** dentro da função dp() e mostre o resultado de **(b)** dentro da função soma().

Exercício – 1 –





- main() chama leitura()
  - leitura() realiza a leitura da matriz 3x3
  - retorna para main()
- main() chama dp()
  - dp() imprime os elementos da diagonal principal
  - retorna para main()
- main() chama soma()
  - soma() realiza a soma entre os elementos que estão abaixo da diagonal principal e imprime o resultado
  - retorna para main()
  - termina o programa.

SOLUÇÃO EM C++

```
#define t 3
int a[t][t]; //Variável Global
void leitura(){
   int i, j; //Variável Local
   for(i = 0; i < t; i++){
      for(j = 0; j < t; j++){
         cout << "\nDigite o elemento [" << i+1 << j+1 << "]: ";
         cin >> a[i][j];
      }
   }
}
void dp(){ //(a)
   int i, j; //Variável Local
   for(i = 0; i < t; i++){
      for(j = 0; j < t; j++){
         if(i == j){
               cout << "\nElemento esta na Diagonal Principal [" << i+1 << j+1 << "]: " << a[i][j];
         }
      }
   }
}
```

```
void soma(){ //(b)
   int i, j, s = 0;
   for(i = 0; i < t; i++){
      for(j = 0; j < t; j++){
         if(i > j){
               s += a[i][j];
         }
      }
   }
   cout << "\nA soma entre os elementos que estão abaixo da Diagonal Principal: " << s;
}

main(){
   leitura();
   dp();
   soma();
}
```

Protótipo de Função:

- É a declaração de uma função que indica o tipo do valor de retorno, a quantidade e o tipo dos argumentos.
- Uma função *não* pode ser chamada antes de ter sido declarada. O *protótipo* de uma função permite que o compilador verifique a sintaxe de sua chamada.

```
#include<iostream>
#include<conio.h>
using namespace std;
#define t 4

void quadrado(); //Protótipo
void leitura(); //Protótipo
int v[t], i;

main(){
    leitura();
    getch();
}
```

```
void quadrado(){
      cout<<"\nO quadrado " << v[i] << ":" << v[i]*v[i];
      cout << endl;
}

void leitura(){
   for (i=0;i<t;i++){
    cout << "\nDigite o elemento " << i+1 << ": ";
    cin >> v[i];
    quadrado();
  }
}
```

Quando utilizamos PROTÓTIPO, as funções são escritas após o main().

Funções que recebem argumentos e retornam valor

```
int soma (int a, int b){
   int s;
   s = a + b;
   return s;
}
```

Funções que recebem argumentos e retornam valor (cont)

```
void leitura();
int soma(int, int);

main()
{
      leitura();
      getch();
}
```

```
int soma(int a, int b){
    int s;
    s = a + b;
    return s;
}
void leitura(){
  int n1, n2;
  cout << "\nDigite o 1o. nr: ";
  cin >> n1;
  cout << "\nDigite o 2o. nr: ";
  cin >> n2;
  cout << "A soma entre os nrs \202: " << soma(n1, n2);
}
```

Os cálculos estão na função
A leitura e a escrita estão no main()

```
#include<iostream>
#include<conio.h>
using namespace std;

int fatorial (int n){
   int i, f=1;
   for (i=1; i<=n; i++){
       f = f * i;
   }
   return f;
}
```

```
main(){
   int x, y;
   cout << "Digite um nr: ";
   cin >> x;
   y = fatorial(x);
   cout << "\nO Fatorial de " << x << " \202: " << y;
   getch();
}
```

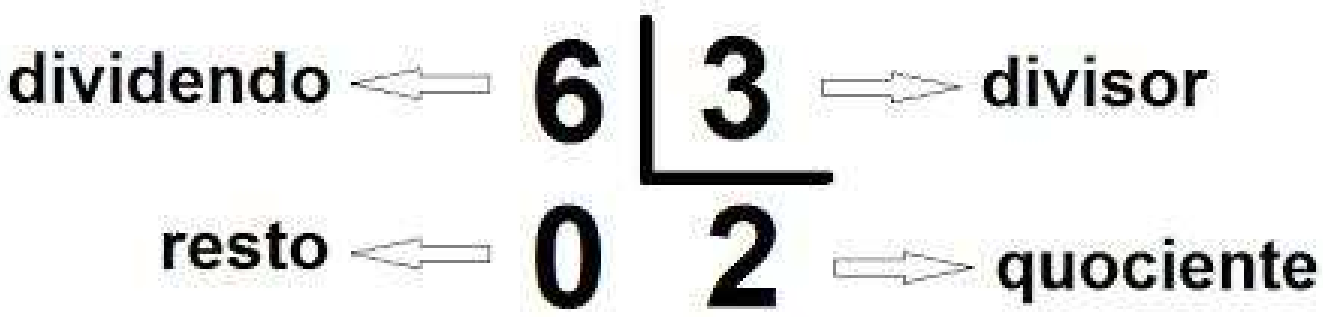
Reutilizando a função fatorial()

```
#include<iostream>
#include<conio.h>
using namespace std;

int fatorial (int n){
   int i, f=1;
   for (i=1; i<=n; i++){
       f = f * i;
   }
   return f;
}
```

```
main(){
   int y;
   y = fatorial(7);
   cout << "\nFat de 7!" << " \202: " << y;
   cout << "\nFat de 5!" << " \202: " << fatorial(5);
   getch();
}
```

Funções que recebem argumentos e retornam valor (cont)



Funções que recebem argumentos e retornam valor (cont)

```
void leitura();
float div(float,float);

main()
{
        leitura();
        getch();
}
```

```
void leitura(){
  float n1, n2;
  cout << "\nDigite o 1o. nr: ";
  cin >> n1;
  cout << "\nDigite o 2o. nr: ";
  cin >> n2;
  cout << "A divisao entre os nrs \202: " << div(n1, n2);
}

float div(float dividendo, float divisor){
    float q;
    q = dividendo / divisor;
    return q;
}
```

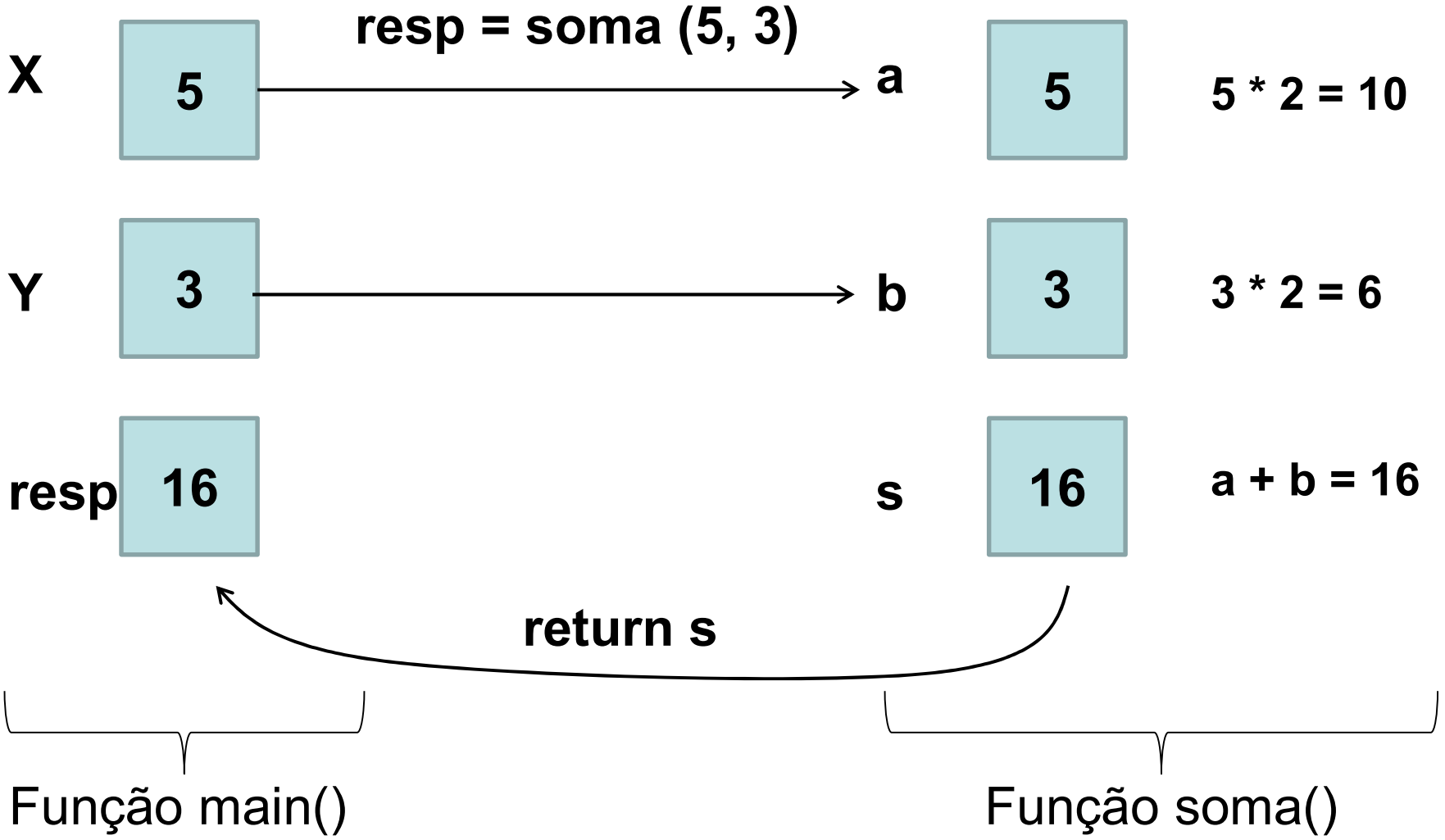
Funções que recebem argumentos e retornam valor (cont)

```
int soma(int,int);

main()
{
    int x, y, resp;
    cout << "\nDigite o 1o. nr: ";
    cin >> x;
    cout << "\nDigite o 2o. nr: ";
    cin >> y;
    resp = soma(x,y);
    cout << "\nA soma: " << resp;
    getch();
}
```

```
int soma(int a, int b){
    int s;
    a = 2 * a;
    b = 2 * b;
    s = a + b;
    return s;
}
```

Funções que recebem argumentos e retornam valor (cont)



Funções que recebem argumentos e retornam valor (cont)

Quando a função *soma* chega ao fim, as variáveis *a*, *b* e *s* são destruídas.

Funções que recebem argumentos e **não** retornam valor

```
void mult(float,int);

main()
{
     int i;
     float v[5], res;
     for(i=0; i<5; i++){
       cout << "\n["<<i+1<<"]: ";
       cin >> v[i];
       mult(v[i],i);
    }
     getch();
}
```

```
void mult(float a, int b){
     float m;
     m = a * b;
     cout << "\nA mult: " << m;
}
```


That's all Folks!