Programação de Computadores

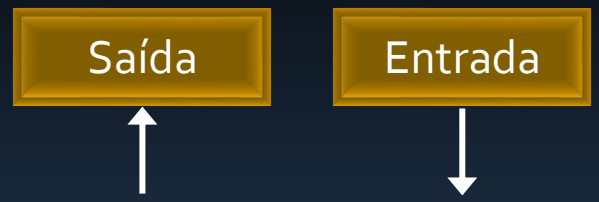
CRIAÇÃO DE FUNÇÕES

Judson Santiago

Introdução

- As funções são importantes para:
  - Dividir o código em blocos
  - Reaproveitar código existente

- A modularização de programas é a principal característica da programação estruturada
  - Facilita a manutenção
  - Encapsula a solução
  - Cria uma interface



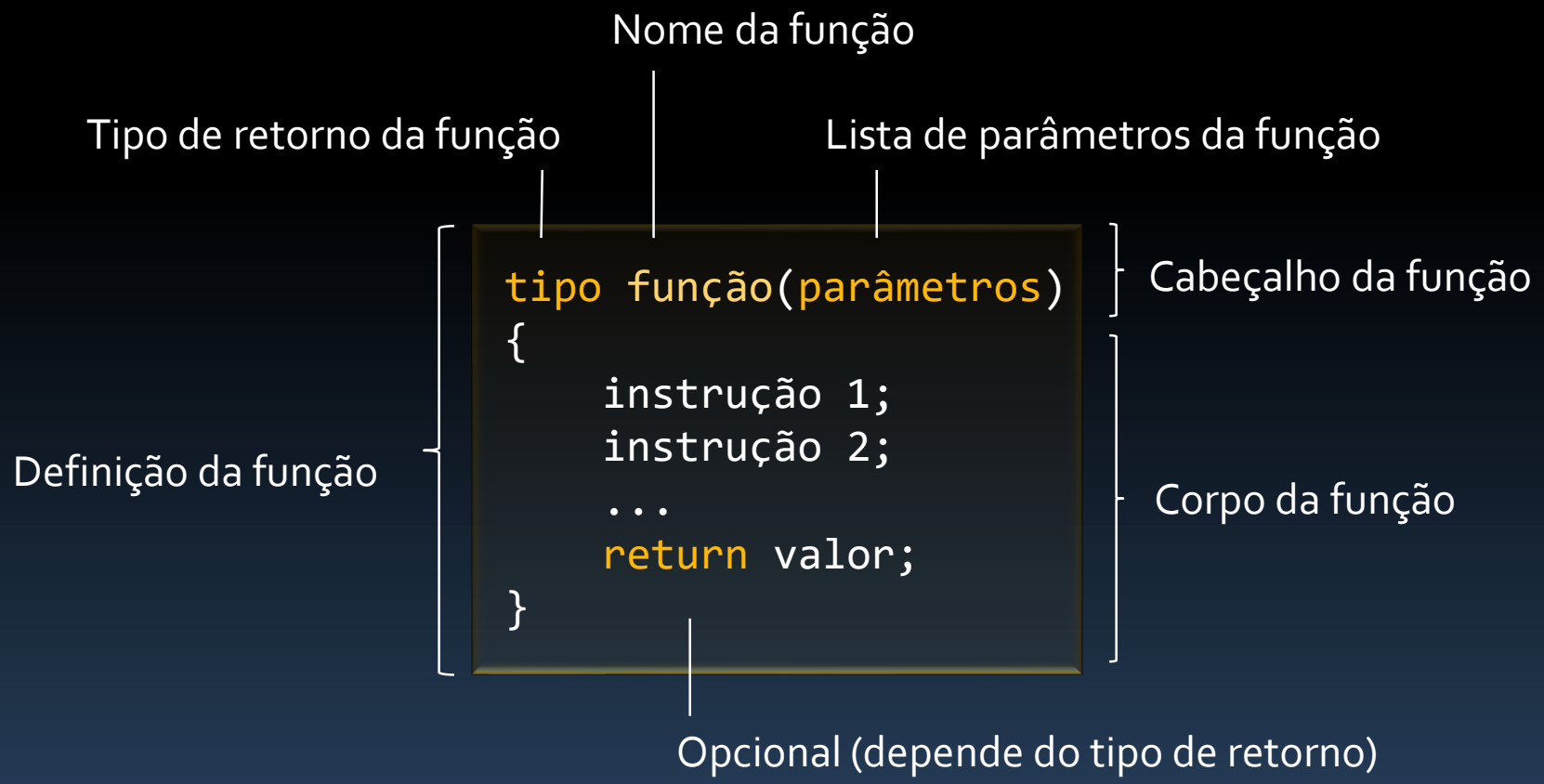
Introdução

- A biblioteca padrão da linguagem C/C++ possui mais de 140 funções predefinidas
  - Prefira utilizar as funções existentes
  - Não reinvente a roda

- Para criar uma função é preciso:
  - Declarar a função (fornecer um protótipo)
  - Definir a função (fornecer um corpo)
  - Chamar a função

Definindo Funções

- A definição de uma função obedece ao modelo:



Funções Sem Retorno

- Funções que não retornam valores são funções de tipo void
  - O retorno é opcional
  - Se o retorno for utilizado ele deve ficar vazio

Tipo de retorno void

```
void função(parâmetros)
{
    instrução 1;
    instrução 2;
    ...
    return;      // opcional
}
```

Funções Sem Retorno

- São normalmente usadas para modularizar o programa

```
void Tchau(int n)
{
    cout << "Finalizando sessão número " << n;
    cout << endl;
}
```

  - Funções sem retorno podem receber valores:

```
int main()
{
     cout << "Encerrar sessão: ";
     int sessao;
     cin >> sessao;
     Tchau(sessao);  // chamada da função
}
```

Funções Com Retorno

- Funções que retornam valores têm a forma geral abaixo:
  - O valor de retorno é obrigatório e pode ser uma constante, uma variável ou uma expressão

Tipo de retorno diferente de void (int, double, char, etc.)

```
tipo função(parâmetros)
{
    instrução 1;
    instrução 2;
    ...
    return valor;
}
```

Funções Com Retorno

- São muito usadas para encapsular cálculos:

```
double Media(double a, double b)
{
    // média aritmética entre a e b
    double m = (a + b)/2;
    return m;
}
```

  - A chamada da função retorna um resultado

```
int main()
{
     double quant;
     quant = Media(12,8); // chamada da função
     cout << "Resultado = " << quant << endl;
}
```

Definindo Funções

- Uma função pode ter vários retornos
  - Mas apenas um deles será executado

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

int main()
{
     cout << "Digite dois números: ";
     int num1, num2;
     cin >> num1 >> num2;
     cout << "O maior deles = " << Maior(num1,num2) << endl;
}
```

Definindo Funções

- Ao contrário de linguagens como Pascal, a linguagem C++ não permite a criação de uma função dentro de outra

```
int main()
{
    void Flexao(int n)
    {
        cout << "Faça " << n << " flexões." << endl;
    }

    Flexao(3);      // chama a função Flexao
    ...
}
```

Parâmetros e Argumentos

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

void Flexao(int);       // protótipo da função Flexao

int main()
{
     Flexao(3);         // chama a função Flexao
     cout << "Escolha um número inteiro: ";
     int cont;
     cin >> cont;
     Flexao(cont);      // chama Flexao novamente
}

void Flexao(int n)
{
     cout << "Faça " << n << " flexões." << endl;
}
```

Flexao.cpp

Parâmetros e Argumentos

- Saída do programa:

```
Faça 3 flexões.
Escolha um número inteiro: 512
Faça 512 flexões.
```

- Uma função pode ser chamada várias vezes
  - A função `Flexao()` é chamada duas vezes:
    - Uma com o argumento sendo o valor 3
    - Outra com o argumento sendo o valor de uma variável

Parâmetros da Função

- Apenas na definição de uma função é preciso dar nome aos parâmetros

```
void Flexao(int);   // protótipo da função
...

void Flexao(int n)  // definição da função
{
     cout << "Faça " << n << " flexões." << endl;
}
```

  - Os parâmetros de uma função são declarações de novas variáveis
    - Elas recebem o valor dos argumentos

Funções e Arquivos

- A definição de várias funções em um arquivo é feita de forma sequencial

Protótipos das funções

```
#include <iostream>

void Flexao(int);
void Abdominal(int);
```

Função Principal

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

Função 1

```
void Flexao(int n)
{
    ...
}
```

Função 2

```
void Abdominal(int n)
{
    ...
}
```

Funções e Arquivos

- Funções podem estar em arquivos diferentes

```
// Arquivo de inclusão:
// ginastica.h

void Flexao(int);
void Abdominal(int);
```

```
// Arquivo fonte:
// ginastica.cpp

void Flexao(int n)
{
   ...
}

void Abdominal(int n)
{
   ...
}
```

```
// Arquivo principal:
// malhando.cpp

#include <iostream>
#include "ginastica.h"

int main()
{
   cout << "Exercícios "
        << "de hoje:"
        << endl;

   Flexao(10);
   Abdominal(20);
}
```

Funções e Arquivos

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

float Media(float, float);

int main()
{
   float a = Media(8,10);
   float b = 12 + Media(15, Media(4,2)) + a;
   cout << "As aulas tem " << b + Media(20,40) << " horas.\n";
}

float Media(float x, float y)
{
   float num = (x + y)/2;
   return num;
}
```

Media.cpp

Funções e Arquivos

- Saída do programa:

```
As aulas tem 60 horas
```

- Uma função pode ser usada como argumento se seu valor de retorno for compatível com o tipo esperado pelo parâmetro

```
float Media(float, float);

float b = 12 + Media( 15 , Media(4,2) ) + a;
```

float float

Inicialização com Funções

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

int Converte(int);   // converte metros em centímetros

int main()
{
     cout << "Entre com a distância em metros: ";
     int num;
     cin >> num;
     int cent = Converte(num);
     cout << num << " metros = " << cent << " centímetros.\n";
}

int Converte(int n)
{
     int val = 100 * n;
     return val;
}
```

Converte.cpp

Inicialização com Funções

- Saída do programa:

```
Entre com a distância em metros: 30
30 metros = 3000 centímetros.
```

- O programa usa cin para obter um valor para a variável num e este valor é passado para a função Converte()

```
cout << "Entre com a distância em metros: ";
cin >> num;

int cent = Converte(num);
```

Retorno de Funções

- Funções com retorno diferente de void devem usar a instrução return para prover o valor de retorno e finalizar a função

```
int main()              // definição da função main
{
      ...
      return 0;
}

int Converte(int n)     // definição da função Converte
{
      ...
      return val;
}
```

Retorno de Funções

- Uma instrução de retorno pode conter uma expressão

```
int Converte(int n)
{
     int val = 100 * n;
     return val;
}
```

Retorno de valor (variável)

```
int Converte(int n)
{
     return 100 * n;
}
```

Retorno de valor (expressão)

Retorno de Funções

- Uma função que retorna valor pode ser usada:
  - No lugar de variáveis
  - No lugar de constantes
  - Em expressões

```
int a = Converte(10);

int b = 20 + Converte(15);

cout << "O tamanho é " << Converte(10) << " centimetros." << endl;
```

Diretiva using com Funções

- A diretiva `using` pode ser usada dentro ou fora da definição

```
#include <iostream>
void flexao(int);

int main()
{
    using namespace std;
    cout << "Escolha um número inteiro: ";
    ...
}

void Flexao(int n)
{
    std::cout << "Faça " << n << " flexões." << std::endl;
}
```

Variável Local vs Global

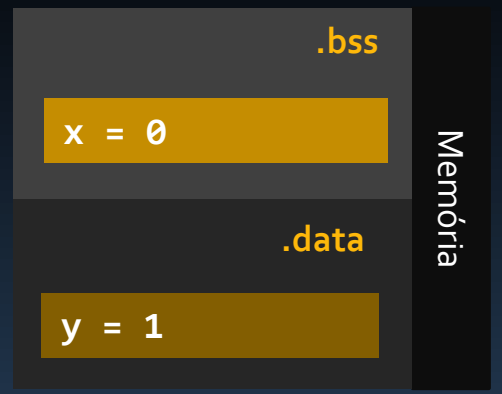
- Uma variável declarada fora de uma função é chamada de **variável global** e visível em todo o código
  - Variáveis globais não inicializadas recebem o valor zero
  - São criadas no início do programa e destruídas no fim

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

int x;      // valor 0
int y = 1;  // valor 1

int main()
{
    ...
```

Região de memória é toda zerada

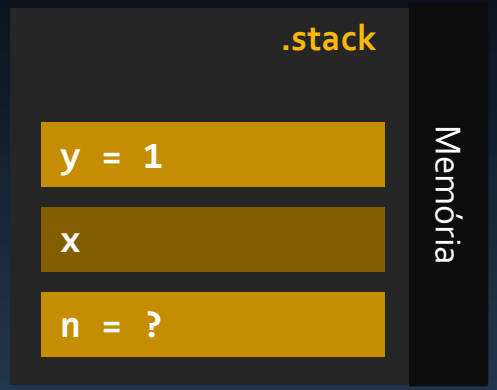


Variável Local vs Global

- Uma variável declarada dentro de uma função é chamada de **variável local** e é visível apenas dentro da função
  - Parâmetros são variáveis locais
  - Não são inicializadas automaticamente para zero
  - São alocadas na entrada e liberadas na saída da função

```
int Converte(int n)
{
      int x;
      int y = 1;
      ...
}
```

Blocos de memória são empilhados



Variável Local vs Global

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

void Dentro();
int x = 1;    // variável global
int y = 2;    // variável global

int main()
{
    cout << "x antes:  " << x << ", y antes:  " << y << endl;
    Dentro();
    cout << "x depois: " << x << ", y depois: " << y << endl;
}

void Dentro()
{
    int y;   // variável local
    x = 3;
    y = 3;
    cout << "x dentro: " << x << ", y dentro: " << y << endl;
}
```

LocalGlobal.cpp

Variável Local vs Global

- Saída do programa:

```
x antes:  1, y antes:  2
x dentro: 3, y dentro: 3
x depois: 3, y depois: 2
```

- A declaração de uma variável local esconde uma variável global de mesmo nome
  - A variável local deixa de existir ao final da função
  - A global se torna visível novamente

Nomes de Funções

- Programadores C++ tem muita flexibilidade na escolha de nomes para funções:

```
MinhaFuncao()    minha_funcao()
minhafuncao()    minha_func()
minhaFuncao()    mf()
```

- Cada programador tem preferência por um estilo
  - Não existe um estilo errado
  - O importante é manter o mesmo padrão em todo o código

Nomes de Funções

- Palavras-chave são o vocabulário de uma linguagem de programação e não podem ser usadas para dar nome a uma função

- Até o momento foram utilizadas as seguintes palavras-chave:

```
int          void         if
double       using        else
float        namespace    return
```

Resumo

- Um programa C++ consiste de uma ou mais funções
- Existem dois tipos de funções
  - Funções que retornam valor: retorno é obrigatório
  - Funções que não retornam valor: tipo void
- Os parâmetros de uma função informam:
  - A quantidade de argumentos
  - Os tipos dos argumentos

```
int Maior(int a, int b);
```

Resumo

- Um programa pode ser quebrado em vários arquivos
  - Arquivos de inclusão (.h): protótipo das funções
  - Arquivos fonte (.cpp): definição das funções
- Variáveis podem ser criadas dentro ou fora de funções
  - Variáveis globais: inicializadas para zero
  - Variáveis locais: contém lixo da memória
- Evite o uso de variáveis globais
  - Dificultam a manutenção do código