BCC 201 - Introdução à Programação I

Procedimentos e Funções

Guillermo Cámara-Chávez
UFOP

Funções e Procedimentos I

- **Procedimentos** são funções estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado.

Funções e Procedimentos II

- **Funções** são procedimentos que retornam um único valor ao final de sua execução.

```
x = sqrt(4);
```

Porque utilizar funções? I

- **Evitar** que os blocos do programa **fiquem grandes demais** (mais difíceis de entender);
- **Facilitar a leitura** do programa-fonte;
- Separar o programa em partes(blocos) que possam ser logicamente **compreendidos de forma isolada**.

Porque utilizar funções? II

- Permitir o **reaproveitamento de código** já construído (por você ou por outros programadores);
- **Evitar** que um trecho de código seja **repetido** várias vezes dentro de um mesmo programa;
- Permitir a **alteração de um trecho** de código de uma forma **mais rápida**.

Declarando uma função? I

- Uma função possui o seguinte formato:

```
tipo nome_da_função (tipo <parametro_1>,
                     tipo <parametro_2>,...,
                     tipo <parametro_n>)
{
    comandos;
    return (valor de retorno);
}
```

Declarando uma função? II

Declarando uma função? III

- Toda função deve ter um tipo.
- Esse tipo determina qual será o tipo de seu **valor de retorno**.
- Os parâmetros de uma função determinam qual será o seu comportamento (como se fosse uma função matemática, onde o parâmetro determina o valor da função).

Declarando uma função? IV

- Uma função **pode não ter parâmetros**, basta não informá-los.
- A expressão contida dentro do comando *return* é chamado de valor de retorno, e corresponde a resposta de uma determinada função.
  - Esse comando é sempre o **último a ser executado** por uma função, e **nada após ele será executado**.
- As funções devem ser **declaradas fora do programa principal** (`main()`).

Declarando uma função? V

- Encontrar o maior de dois números

Declarando uma função? VI

```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Declarando uma função? VII

```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Declarando uma função? VIII

```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Declarando uma função? IX

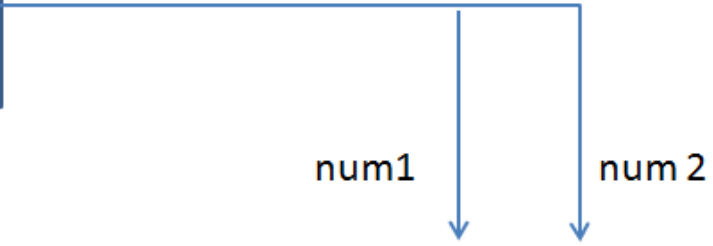
```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Declarando uma função? X

```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Declarando uma função? XI

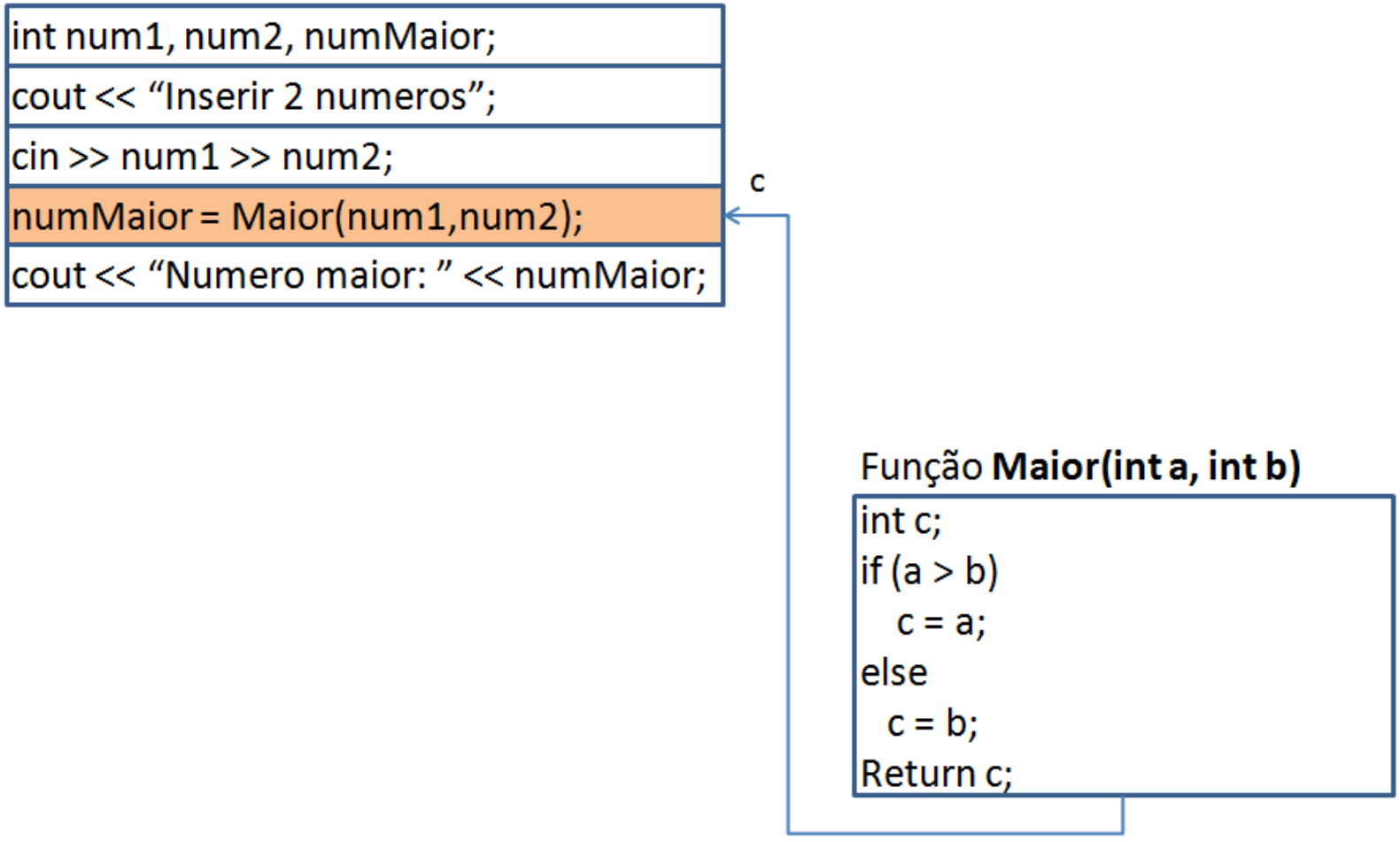
```
int num1, num2, numMaior;
cout << "Inserir 2 numeros";
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << "Numero maior: " << numMaior;
```



Função **Maior(int a, int b)**

```
int c;
if (a > b)
   c = a;
else
  c = b;
Return c;
```

Declarando uma função? XII



Declarando uma função? XIII

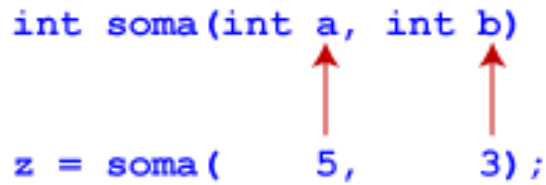
```
int num1, num2, numMaior;
cout << “Inserir 2 numeros”;
cin >> num1 >> num2;
numMaior = Maior(num1,num2);
cout << “Numero maior: ” << numMaior;
```

Exemplo de função I

A função abaixo soma dois valores passados como parâmetro

```
int soma (int a, int b)
{
    int c = 0;
    c = a + b;
    return (c);
}
```

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



Invocando uma função I

Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável.

```
variavel = funcao (parametros);
```

Podemos invocar uma função em qualquer lugar onde faríamos a leitura de uma variável, mas nunca a escrita. Ex:

```
cout << "Soma de a e b : " << soma(a,b);
```

O que podemos entender por escrita?

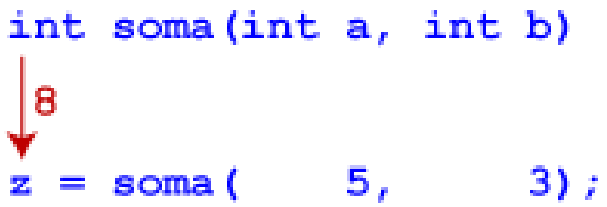
Invocando uma função II

Na seguinte instrução é realizado a **leitura** do valor de y e logo atribuído à variável x, neste ponto é executado a **escrita** do valor de y em x.

```
x = y;
```

Vejamos o seguinte programa. A terceira instrução está errada.

```
int a = 5, b = 3, z;
z = soma(a, b);
soma(a,b) = 8;   // ERRADO!
```



Invocando uma função III

- As variáveis passadas como **parâmetros indicam** quais são os **valores com os quais a função irá trabalhar**.
- Esses **valores são copiados** para os parâmetros da função, que pode manipulá-los.
- **Os parâmetros passados pela função não necessariamente possuem os mesmos nomes** que os parâmetros que a função espera.

Invocando uma função IV

```
int soma(int a, int b){
    return (a+b);
}
int main(){
    int x = 10, y = 30, z;
    z = soma(x, y);
    cout << "Soma de " << x << " e "
         << y << " = " << z;
    //Está correto??
    cout << "Soma de " << a << " e "
         << b << " = " << z;
}
```

Invocando uma função V

```
int soma(int a, int b){
    return (a+b);
}
int main(){
    int x = 10, y = 30, z;
    z = soma(x, y);
    cout << "Soma de " << x << " e "
         << y << " = " << z;
    //Está correto??
    cout << "Soma de " << a << " e "
         << b << " = " << z;
}
```

O segundo cout está errado. As variáveis *a* e *b* são variáveis privadas que somente pertencem a função e não tem como ter acesso a elas dentro do programa principal.

O tipo *void*

- Tipo especial, utilizado principalmente em funções
- Representa o "nada"
  - uma variável desse tipo armazena conteúdo indeterminado e
  - uma função desse tipo retorna um conteúdo indeterminado.

Procedimento em C I

- Os procedimentos em linguagem C são funções do tipo `void`.
- O procedimento abaixo imprime um número que é passado como parâmetro

```
void imprime (int numero)
{
    cout<< "Numero = " << numero;
}
```

Podemos ignorar o valor de retorno de uma função e, para esta chamada, ela será equivalente a um procedimento.

A função *main*

- O programa principal na verdade é uma função, ele é uma função especial.
- Ele é invocada automaticamente pelo programa quando esse inicia sua execução e possui um tipo fixo (`int`).
- O comando return informa ao sistema operacional se o programa funcionou corretamente ou não.
- O padrão é que um programa retorne zero caso tenha funcionado corretamente ou qualquer outro valor caso contrário.

Declarando uma função I

- Declarar uma função sem a sua implementação é como declará-la com a implementação
- Para declará-la sem a implementação basta substituir as chaves e seu conteúdo por ponto e vírgula.

```
tipo nome_da_funcao (tipo <parametro_1>,
                     tipo <parametro_2>, ...,
                     tipo <parametro_n>);
```

Declarando uma função II

```
#include <stdio.h>
int soma(int a, int b);
int main()
{
    int c, x, y;
    cout << "Inserir dois números \n";
    cin >> x >> y;
    cout << "A soma eh " << soma(x,y);
}
int soma(int a, int b)
{
    return (a+b);
}
```

Exemplos I

- Crie uma função para determinar se um número é ou não par.

Exemplos II

```
int EhPar(int num);
int main()
{
    int x;
    cout << "Inserir um número \n";
    cin >> x;
    if (EhPar(x) == 1)
        cout << "Numero par! \n";
    else
        cout << "Numero nao eh par! \n";
}
int EhPar(int num)
{
    if (num % 2 == 0)
        return 1; // Verdadeiro
    else
        return 0; // Falso
}
```

Exemplos III

- Leia dois catetos e (enquanto não for digitado zero na leitura do primeiro cateto) mostre a hipotenusa. Crie, além da função para calcular a hipotenusa, o procedimento para mostrar o valor da hipotenusa.

Exemplos IV

```
double Pitagoras (double cat1, double cat2);
void Mostrar(double hip);
int main()
{
    double cat1, cat2, hip;
    while(1)
    {
        cout << "Inserir primeiro cateto
                  (0 para finalizar)\n";
        cout << cat1;
        if (cat1 == 0) break;
        cout << "Inserir segundo cateto\n";
        cin >> cat2;
        hip = Pitagoras(cat1, cat2);
        Mostrar(hip);
    }
    return 0;
}
```

Exemplos V

```
double Pitagoras (double cat1, double cat2)
{
    return sqrt(pow(cat1,2) + pow(cat2,2));
}
void Mostrar(double hip)
{
    cout << "Hipotenusa: " << hip;
}
```

Exemplos VI

- Determine se um número inteiro é ou não primo, criando uma função para isso. Sair com zero.

Exemplos VII

```
#include <stdio.h>
int EhPrimo(int numero);
int main()
{
    int num, primo;
    while(1)
    {
        cout << "Inserir numero
                  (0 para finalizar)\n";
        cin >> num;
        if (num == 0) break;
        if (EhPrimo(num) == 1)
            cout << "Numero eh primo! \n";
        else
            cout << "Numero nao eh primo! \n";
    }
    return 0;
}
```

Exemplos VIII

```
int EhPrimo(int numero)
{
    int cont = 0, i;
    for (i = 1; i <= numero; i++)
    {
        if (numero % i == 0)
            cont++;
    }
    if (cont == 2)
        return 1; // Verdadeiro
    else
        return 0; // Falso
}
```

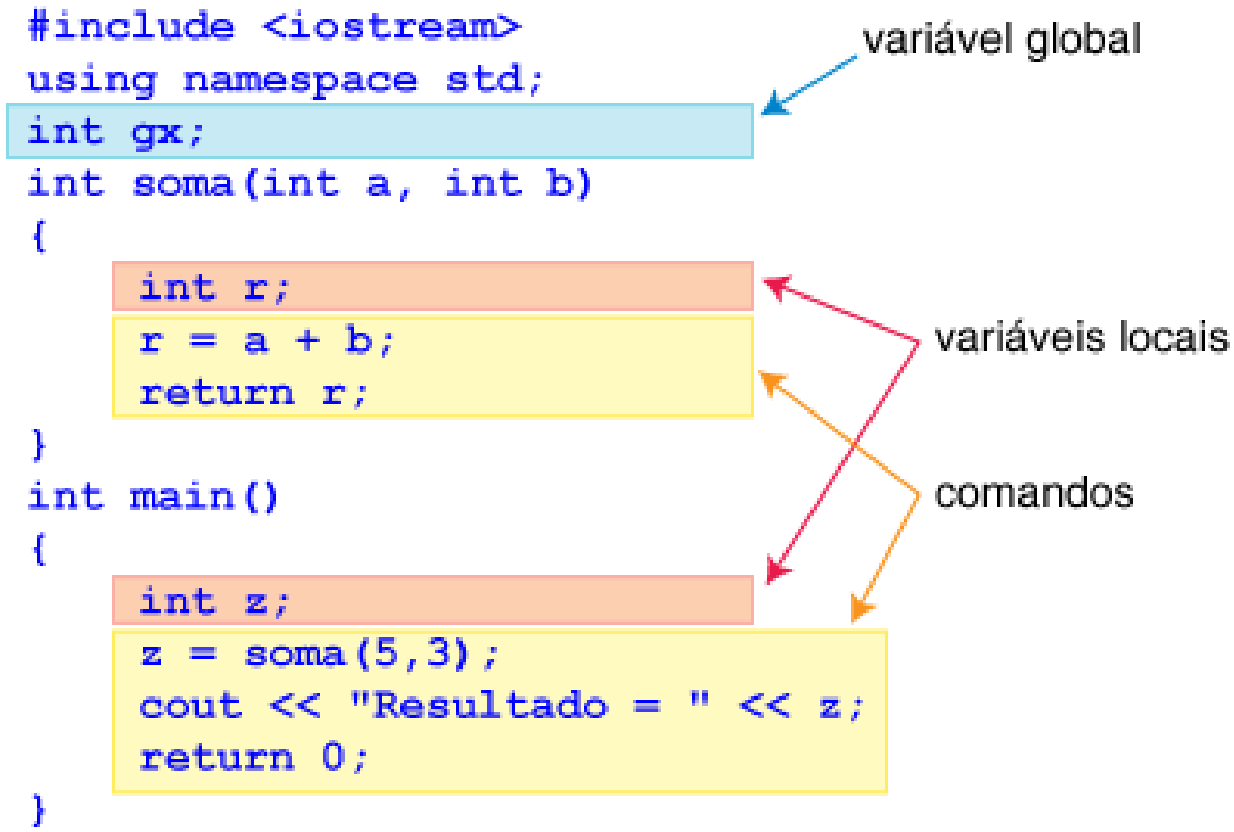
Variáveis locais e variáveis globais I

- Variável local
  - é declarada dentro de uma função,
  - ela existe somente dentro de uma função,
  - após o término da execução da função, ela deixa de existir.

Variáveis locais e variáveis globais II

- Variável global
  - ela é declarada fora da função (ou seja, no mesmo lugar onde registros (*struct*), tipos enumerados (*enum*) e procedimentos são declarado)
  - Existe dentro de todas as funções e qualquer procedimento/função pode alterá-las.

Variáveis locais e variáveis globais III



Variáveis locais e variáveis globais IV

- Ex. variável global:

```
#include <iostream>

int var_global = 10;
void IncrementaGlobal(int a);

int main(){
    int i;
    cin >> i;
    cout << "variavel global eh " << var_global;
    IncrementaGlobal(i);
    cout << "variavel global eh " << var_global;
}

void IncrementaGlobal(int a){
    var_global += a; //var_global = var_global + a
}
```

Variáveis locais e variáveis globais V

- **Notas:**
  - evite utilizar variáveis globais
  - variáveis globais são sinais de um mau *design*
  - existe o problema que duas funções modifiquem o valor da variável no mesmo instânte

Variáveis locais e variáveis globais VI

Outro exemplo:

```
#include <iostream>

int var_global = 10;
void MostraGlobal_e_Local(int, int);

int main()
{
    int i, j;
    cout << "Inserir dois numero:";
    cin >> i >> j;
    MostraGlobal_e_Local(i, j);
    cout << "Main->variavel global eh: "
         << var_global << endl;
    cout << "Main->variavel local eh "
         << i << endl;
    return 0;
}
```

Variáveis locais e variáveis globais VII

```
void MostraGlobal_e_Local(int var_global, int num)
{
    cout << "Funcao->Privada: "
         << var_global << endl;
    cout << "Funcao->Outra Privada: "
         << num << endl;
    cout << "Funcao->GlobalPrivada: "
         << ::var_global << endl;
    var_global += 5;
    num += 5;
    ::var_global += 5;
}
```

Variáveis locais e variáveis globais VIII

Seria mostrado na tela:

```
Inserir dois numero:50
60
Funcao->Privada: 50
Funcao->Outra Privada: 60
Funcao->GlobalPrivada: 10

Main->variavel global eh: 15
Main->variavel local eh: 50
```

FIM