9002 — Aula 13
Algoritmos e Programação de Computadores

Instituto de Engenharia – UFMT

Segundo Semestre de 2014

10 de novembro de 2014

Roteiro

Revisão

Na aula anterior...

- vimos como organizar melhor nossos códigos com funções.
- vimos como definir e invocar funções.
- aprendemos como simular códigos etc

Nesta aula, aprofundaremos um pouco mais nosso conhecendo sobre este tema. Vamos lá...

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

Funções e procedimentos

- Funções podem apenas retornar um valor
- Também podem apenas realizar um procedimento

Função

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

Procedimento

```
void soma(int a, int b) {
    printf("Soma:%d", a+b);
}
```

Procedimentos

- Um procedimento é uma função que não retorna nenhum valor.
- Procedimentos são indicados pela palavra-chave especial **void**.

A palavra-chave void

- Em C, a palavra `void` representa a "ausência" de algum objeto. Pode ser usado para
  - indicar a ausência de um tipo de retorno,
  - explicitar a ausência de parâmetros em uma função.

Sem tipo de retorno

```
void imprime_numero(int n) {
    printf("Num:  %d\n", n);
}
```

Sem parâmetros

```
int obtem_numero(void) {
    int n;
    scanf("%d", &n);
    return n;
}
```

A palavra-chave void

- Em C, a palavra `void` representa a "ausência" de algum objeto. Pode ser usado para
  - indicar a ausência de um tipo de retorno,
  - explicitar a ausência de parâmetros em uma função.

Sem tipo de retorno

```
void imprime_numero(int n) {
    printf("Num:  %d\n", n);
}
```

Sem parâmetros

```
int obtem_numero(void) {
    int n;
    scanf("%d", &n);
    return n;
}
```

A palavra-chave void

- Em C, a palavra `void` representa a "ausência" de algum objeto. Pode ser usado para
  - indicar a ausência de um tipo de retorno,
  - explicitar a ausência de parâmetros em uma função.

Sem tipo de retorno

```
void imprime_numero(int n) {
    printf("Num:  %d\n", n);
}
```

Sem parâmetros

```
int obtem_numero(void) {
    int n;
    scanf("%d", &n);
    return n;
}
```

A função main

- O programa principal é uma função especial com tipo de retorno `int`
- A função `main` é invocada automaticamente pelo sistema operacional
- O valor retornado deve valer:
  - zero caso tenha funcionado corretamente
  - qualquer outro valor caso tenha ocorrido um erro.

Exemplo

```
int main() {
    float dividendo, divisor;
    scanf("%f %f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao por zero!");
        return 1;
    } else {
        printf("Divisao: %f.", dividendo / divisor);
        return 0;
    }
}
```

A função main

- O programa principal é uma função especial com tipo de retorno `int`
- A função `main` é invocada automaticamente pelo sistema operacional
- O valor retornado deve valer:
  - zero caso tenha funcionado corretamente
  - qualquer outro valor caso tenha ocorrido um erro.

Exemplo

```
int main() {
    float dividendo, divisor;
    scanf("%f %f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao por zero!");
        return 1;
    } else {
        printf("Divisao: %f.", dividendo / divisor);
        return 0;
    }
}
```

A função main

- O programa principal é uma função especial com tipo de retorno `int`
- A função `main` é invocada automaticamente pelo sistema operacional
- O valor retornado deve valer:
  - zero caso tenha funcionado corretamente
  - qualquer outro valor caso tenha ocorrido um erro.

Exemplo

```
int main() {
    float dividendo, divisor;
    scanf("%f %f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao por zero!");
        return 1;
    } else {
        printf("Divisao: %f.", dividendo / divisor);
        return 0;
    }
}
```

A função main

- O programa principal é uma função especial com tipo de retorno `int`
- A função `main` é invocada automaticamente pelo sistema operacional
- O valor retornado deve valer:
  - zero caso tenha funcionado corretamente
  - qualquer outro valor caso tenha ocorrido um erro.

Exemplo

```
int main() {
    float dividendo, divisor;
    scanf("%f %f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao por zero!");
        return 1;
    } else {
        printf("Divisao: %f.", dividendo / divisor);
        return 0;
    }
}
```

A função main

- O programa principal é uma função especial com tipo de retorno `int`
- A função `main` é invocada automaticamente pelo sistema operacional
- O valor retornado deve valer:
  - zero caso tenha funcionado corretamente
  - qualquer outro valor caso tenha ocorrido um erro.

Exemplo

```
int main() {
    float dividendo, divisor;
    scanf("%f %f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao por zero!");
        return 1;
    } else {
        printf("Divisao: %f.", dividendo / divisor);
        return 0;
    }
}
```

Declarando funções depois do `main`

Considere o seguinte código.

```
#include <stdio.h>

int main () {
    float a = 0, b = 5;
    printf("%f\n", soma(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return op1 + op2;
}
```

Ocorre um erro de compilação! Por quê?

Declarando funções depois do `main`

Considere o seguinte código.

```
#include <stdio.h>

int main () {
    float a = 0, b = 5;
    printf("%f\n", soma(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return op1 + op2;
}
```

Ocorre um erro de compilação! Por quê?

Protótipos de funções

- Para organizar melhor um programa e implementar funções em partes distintas do arquivo são utilizados **protótipos de funções**.
- Protótipos de funções correspondem à primeira linha da declaração de uma função contendo tipo de retorno, nome da função, parâmetros e **um ponto e vírgula no final**.

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve vir sempre antes do seu uso. A sua definição pode aparecer em qualquer lugar do programa.
- É comum sempre colocar os protótipos de funções no início do seu arquivo.

Protótipos de funções

- Para organizar melhor um programa e implementar funções em partes distintas do arquivo são utilizados **protótipos de funções**.
- Protótipos de funções correspondem à primeira linha da declaração de uma função contendo tipo de retorno, nome da função, parâmetros e **um ponto e vírgula no final**.

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve vir sempre antes do seu uso. A sua definição pode aparecer em qualquer lugar do programa.
- É comum sempre colocar os protótipos de funções no início do seu arquivo.

Protótipos de funções

- Para organizar melhor um programa e implementar funções em partes distintas do arquivo são utilizados **protótipos de funções**.
- Protótipos de funções correspondem à primeira linha da declaração de uma função contendo tipo de retorno, nome da função, parâmetros e **um ponto e vírgula no final**.

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve vir sempre antes do seu uso. A sua definição pode aparecer em qualquer lugar do programa.
- É comum sempre colocar os protótipos de funções no início do seu arquivo.

Protótipos de funções

- Para organizar melhor um programa e implementar funções em partes distintas do arquivo são utilizados **protótipos de funções**.
- Protótipos de funções correspondem à primeira linha da declaração de uma função contendo tipo de retorno, nome da função, parâmetros e **um ponto e vírgula no final**.

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve vir sempre antes do seu uso. A sua definição pode aparecer em qualquer lugar do programa.
- É comum sempre colocar os protótipos de funções no início do seu arquivo.

Protótipo de funções

```
#include <stdio.h>

float soma (float op1, float op2);

int main () {
    float a = 0, b = 5;
    printf ("%f\n", soma (a, b));
    return 0;
}

float soma (float op1, float op2) {
    return (op1 + op2);
}
```

Protótipo de funções

```
#include <stdio.h>

float soma (float op1, float op2);
float subt (float op1, float op2);

int main () {
    float a = 0, b = 5;
    printf ("%f\n  %f\n", soma (a, b), subt(a, b));
    return 0;
}

float soma (float op1, float op2) {
    return (op1 + op2);
}

float subt (float op1, float op2) {
    return (op1 - op2);
}
```

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Variáveis locais e variáveis globais

Variável local

Uma variável é chamada local se ela foi declarada dentro de uma função.

- A variável existe somente dentro da função
- Quando a execução da função termina, a variável deixa de existir
- Parâmetros são variáveis locais

Variável global

Uma variável é chamada global se ela for declarada fora de qualquer função.

- A variável é visível em todas as funções
- Qualquer função pode modificar a variável
- A variável existe durante toda a execução do programa
- Só deve ser usada em casos muito especiais

Revisando a estrutura básica de um programa

```
#include <stdio.h>

Protótipos de funções
Declaração de variáveis globais

int main() {
    Declaração de variáveis locais
    Comandos
}

Outras funções

int exemplo_funcao(float parametro, int outro) {
    Declaração de variáveis locais
    Comandos
}
```

Escopo de variáveis

- O escopo de uma variável determina onde ela pode ser acessada.
- A regra de escopo em C é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

- O escopo de uma variável determina onde ela pode ser acessada.
- A regra de escopo em C é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis - Exemplo

```
#include <stdio.h>

void funcao_a(void);
int  funcao_b(int local_b);

int global;

int main() {
    int local_main;
    /* Neste ponto são visíveis global e local_main */
}

void funcao_a(void) {
    int local_a;
    /* Neste ponto são visíveis global e local_a */
}

int funcao_b(int local_b){
    int local_c;
    /* Neste ponto são visíveis global, local_b e local_c */
}
```

Escopo de variáveis - Escondendo variáveis globais

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local "esconde" a variável global.

```
int nota;

void a() {
    int nota;
    /* Neste ponto nota é a variável local. */
}
```

Escopo de variáveis - Escondendo variáveis globais

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local "esconde" a variável global.

```
int nota;

void a() {
    int nota;
    /* Neste ponto nota é a variável local. */
}
```

Escopo de variáveis - Escondendo variáveis globais

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local "esconde" a variável global.

```
int nota;

void a() {
    int nota;
    /* Neste ponto nota é a variável local. */
}
```

Escopo de variáveis - Escondendo variáveis globais

```
#include <stdio.h>

int x;

void fun1(){
    printf("\n%d",x);
}

void fun2(){
    int x;
    printf("\n%d",x);
}

int main(){
    x = 10;
    fun1();
}
```

O que é impresso na chamada de fun1? E se fosse chamada fun2?

Escopo de variáveis - Escondendo variáveis globais

```
#include <stdio.h>

int x;

void fun1(){
    printf("\n%d",x);
}

void fun2(){
    int x;
    printf("\n%d",x);
}

int main(){
    x = 10;
    fun1();
}
```

O que é impresso na chamada de fun1? E se fosse chamada fun2?

Escopo de variáveis - Escondendo variáveis globais

```
#include <stdio.h>

int x;

void fun1(){
    printf("\n%d",x);
}

void fun2(){
    int x;
    printf("\n%d",x);
}

int main(){
    x = 10;
    fun1();
}
```

O que é impresso na chamada de fun1? E se fosse chamada fun2?

Passagem de parâmetros

Considere o seguinte código:

```
void trocar_valores(int x, int y) {
    int aux;
    aux = x;
    x = y;
    y = aux;
}

int main(){
    int x = 4, y = 5;
    trocar_valores(x, y);
    printf("x=%d, y=%d", x, y);
}
```

O que é impresso? Por que não funcionou a troca?

Passagem de parâmetros

Considere o seguinte código:

```
void trocar_valores(int x, int y) {
    int aux;
    aux = x;
    x = y;
    y = aux;
}

int main(){
    int x = 4, y = 5;
    trocar_valores(x, y);
    printf("x=%d, y=%d", x, y);
}
```

O que é impresso? Por que não funcionou a troca?

Passagem de parâmetros

Considere o seguinte código:

```
void trocar_valores(int x, int y) {
    int aux;
    aux = x;
    x = y;
    y = aux;
}

int main(){
    int x = 4, y = 5;
    trocar_valores(x, y);
    printf("x=%d, y=%d", x, y);
}
```

O que é impresso? Por que não funcionou a troca?

Passagem de parâmetros

- Os parâmetros de função podem se passados de dois modos.
  1. Por valor: valores são copiados **(Esse é o modo padrão)**
  2. Por referência: uma referência para a variável é passada

Por valor

- Apenas o resultado da expressão é passado
- Pode ser qualquer expressão: constantes, somas, etc.
- Os valores são atribuídos para os parâmetros formais
- Não alteram o valor das variáveis passadas

Por referência

- Uma referência de uma variável é passada
- Apenas variáveis podem ser utilizadas
- A referência deve ser explícita com o operador &
- Alteram o valor das variáveis passadas

Passagem de parâmetros

- Os parâmetros de função podem se passados de dois modos.
  1. Por valor: valores são copiados **(Esse é o modo padrão)**
  2. Por referência: uma referência para a variável é passada

Por valor

- Apenas o resultado da expressão é passado
- Pode ser qualquer expressão: constantes, somas, etc.
- Os valores são atribuídos para os parâmetros formais
- Não alteram o valor das variáveis passadas

Por referência

- Uma referência de uma variável é passada
- Apenas variáveis podem ser utilizadas
- A referência deve ser explícita com o operador &
- Alteram o valor das variáveis passadas

Passagem de parâmetros

- Os parâmetros de função podem se passados de dois modos.
  1. Por valor: valores são copiados **(Esse é o modo padrão)**
  2. Por referência: uma referência para a variável é passada

Por valor

- Apenas o resultado da expressão é passado
- Pode ser qualquer expressão: constantes, somas, etc.
- Os valores são atribuídos para os parâmetros formais
- Não alteram o valor das variáveis passadas

Por referência

- Uma referência de uma variável é passada
- Apenas variáveis podem ser utilizadas
- A referência deve ser explícita com o operador &
- Alteram o valor das variáveis passadas

Passagem de parâmetros

- Os parâmetros de função podem se passados de dois modos.
  1. Por valor: valores são copiados **(Esse é o modo padrão)**
  2. Por referência: uma referência para a variável é passada

Por valor

- Apenas o resultado da expressão é passado
- Pode ser qualquer expressão: constantes, somas, etc.
- Os valores são atribuídos para os parâmetros formais
- Não alteram o valor das variáveis passadas

Por referência

- Uma referência de uma variável é passada
- Apenas variáveis podem ser utilizadas
- A referência deve ser explícita com o operador &
- Alteram o valor das variáveis passadas

Passagem de parâmetros

- Os parâmetros de função podem se passados de dois modos.
  1. Por valor: valores são copiados **(Esse é o modo padrão)**
  2. Por referência: uma referência para a variável é passada

Por valor

- Apenas o resultado da expressão é passado
- Pode ser qualquer expressão: constantes, somas, etc.
- Os valores são atribuídos para os parâmetros formais
- Não alteram o valor das variáveis passadas

Por referência

- Uma referência de uma variável é passada
- Apenas variáveis podem ser utilizadas
- A referência deve ser explícita com o operador &
- Alteram o valor das variáveis passadas

Passagem por valor

```
int obter_quadrado(int x) {
    return x*x;
}

int main() {
    int x = 2;
    printf("Quadrado de %d eh %d\n", x, obter_quadrado(x));
    printf("Quadrado de %d eh %d\n", 3, obter_quadrado(3));
}
```

Passagem por referência

- Obtemos uma referência para a variável com o operador &
- Para acessar a variável referenciada dentro da função devemos preceder a variável com o símbolo *

```
void calcular_quadrado(int x, int *quadrado) {
    *quadrado = x*x;
}


int main() {
    int x = 2, quadrado;
    calcular_quadrado(x, &quadrado);
    printf("Quadrado de %d eh %d\n", x, quadrado);
}
```

Passagem por referência

- Obtemos uma referência para a variável com o operador &
- Para acessar a variável referenciada dentro da função devemos preceder a variável com o símbolo *

```
void calcular_quadrado(int x, int *quadrado) {
    *quadrado = x*x;
}

int main() {
    int x = 2, quadrado;
    calcular_quadrado(x, &quadrado);
    printf("Quadrado de %d eh %d\n", x, quadrado);
}
```

Ponteiros

Ponteiros

Em C, referências são implementadas por meio de ponteiros. Ponteiros correspondem ao endereço da variável na memória.

Regras:

- Declaramos um ponteiro como *tipo *ponteiro*;
- Obtemos um ponteiro de uma variável com o operador &
- Obtemos a variável de um ponteiro com o operador *

Ponteiros

Ponteiros

Em C, referências são implementadas por meio de ponteiros. Ponteiros correspondem ao endereço da variável na memória.
Regras:

- Declaramos um ponteiro como *tipo* ***ponteiro***;
- Obtemos um ponteiro de uma variável com o operador &
- Obtemos a variável de um ponteiro com o operador *

Ponteiros

Ponteiros

Em C, referências são implementadas por meio de ponteiros. Ponteiros correspondem ao endereço da variável na memória.
Regras:

- Declaramos um ponteiro como *tipo *ponteiro*;
- Obtemos um ponteiro de uma variável com o operador &
- Obtemos a variável de um ponteiro com o operador *

Ponteiros

Ponteiros

Em C, referências são implementadas por meio de ponteiros. Ponteiros correspondem ao endereço da variável na memória.
Regras:

- Declaramos um ponteiro como *tipo* ***ponteiro***;
- Obtemos um ponteiro de uma variável com o operador &
- Obtemos a variável de um ponteiro com o operador *

Ponteiros - Exemplo

Ponteiros

```
int variavel;
int *ponteiro1, *ponteiro2;

variavel = 3;

ponteiro1 = &variavel;
ponteiro2 = ponteiro1;

*ponteiro2 = 5;

printf("%d, %d, %d", variavel, *ponteiro1, *ponteiro2);
```

Irá imprimir: 5, 5, 5.

Ponteiros - Exemplo

Ponteiros

```
int variavel;
int *ponteiro1, *ponteiro2;

variavel = 3;

ponteiro1 = &variavel;
ponteiro2 = ponteiro1;

*ponteiro2 = 5;

printf("%d, %d, %d", variavel, *ponteiro1, *ponteiro2);
```

Irá imprimir: 5, 5, 5.

Exercícios

1. Escreva uma função que troca os valores de duas variáveis inteiras.
2. Escreva uma função que decida se um número é produto de dois números ímpares. Se for, a função deverá retornar esses dois números em variáveis passadas por referência.
3. Escreva uma função que decida se um número é produto de quatro números ímpares. Se for, a função deverá retornar os números em variáveis passadas por referência. Tente utilizar a função anterior.

Nas próximas aulas...

- Na próxima aula, resolveremos mais exercícios.
- FIM!