

Algoritmos e Programação de Computadores

Disciplina 113476

Prof. Alexandre Zaghetto
<http://alexandre.zaghetto.com>
zaghetto@unb.com



<http://www.nickgentry.com/>

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

O presente conjunto de *slides* não pode ser reutilizado ou republicado sem a permissão do instrutor.

Módulo 11

Subalgoritmos

(Funções)

1. Funções

- Frequentemente temos de desenvolver programas para resolver problemas que necessitam de algoritmos extensos e complexos.
- Isso costuma implicar em códigos mais difíceis de ler e em repetição de trechos de código ao longo do programa.
- Imagine uma fábrica de automóveis. Ela projeta e monta os carros, mas não fabrica os pneus e os vidros, por exemplo.
- Ela os compra pronto ou produz em outros lugares, terceirizando esse trabalho, pois seria complicado manter uma linha de montagem de veículos, uma fábrica de pneus e uma de vidros juntas e gerir isso tudo.

1. Funções

- Em programação podemos fazer um processo análogo, “terceirizando” a solução de um pedaço de nosso problema ao passar o trabalho para um módulo específico do programa.
- Em programação podemos dividir os **algoritmos** em **subalgoritmos** menores e de mais fácil compreensão.
- Com isso, dividimos um problema difícil em vários problemas mais fáceis, juntando tudo no final e formando uma solução completa.

1. Funções

- Até agora, em todos os programas que criamos, codificamos uma única função: a função ***main()***.
- Entretanto, em todos eles, diversas funções foram utilizadas: ***system()***, ***printf()***, ***scanf()***, ***getch()***, ***putch()***, ***sqrt()***, ***pow()*** etc.
- Essas funções estão disponíveis no sistema através de bibliotecas que acompanham o compilador C.
- Mas podemos definir nossas próprias funções e utilizá-las da mesma maneira.



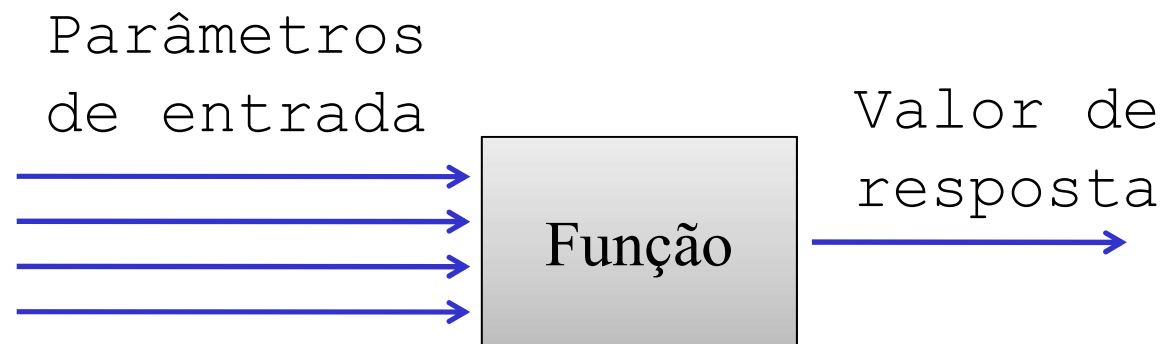
2. Definindo Funções

```
tipo <nome> (<parâmetros>) {  
    <Declarações de variáveis>  
    <comandos>  
}
```

- **tipo** refere-se ao tipo de resposta que a função devolve e deve ser **void** (vazio) se a função não tem valor de resposta;
- **nome** é o identificador da função no resto do programa;
- **parâmetros** é uma lista de variáveis que representam valores de entrada para a função e deve ser **void** caso não haja valores de entrada;
- Dentro do corpo da função, a primeira seção é destinada à declaração das variáveis e a segunda, aos comandos.

2. Definindo Funções

- Caso geral.



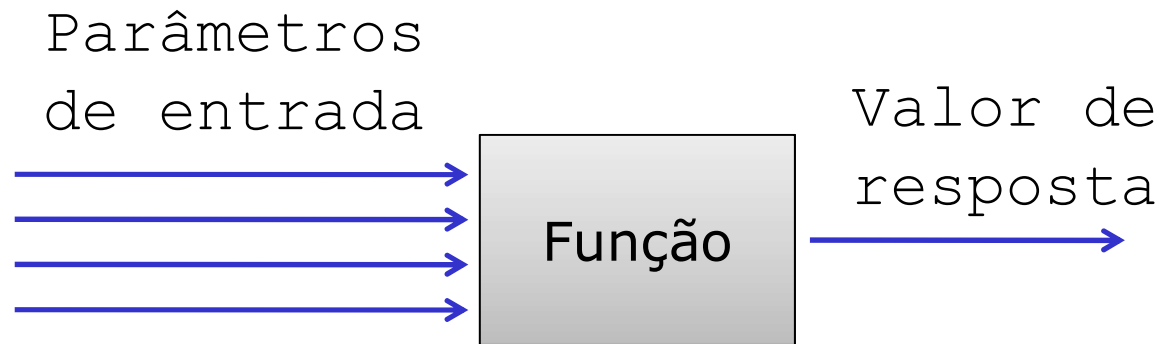
2. Definindo Funções

- Função que não tem valor de resposta e não recebe argumentos ao ser chamada.

Exemplo: Escreva uma função que imprime na tela do computador um cabeçalho com a identificação do aluno, contendo seu nome, matrícula e curso. Escreva também um programa principal que utiliza a função desenvolvida.

2. Definindo Funções

- Função que não tem valor de resposta e não recebe argumentos ao ser chamada.



2. Definindo Funções

- Função que não tem valor de resposta e não recebe argumentos ao ser chamada.



Função

}



2. Definindo Funções

```
void indentifica_aluno(void) {
```



Definição da função.

```
    printf("Nome: Alexandre Zaghetto \n");  
    printf("Matricula: 123456 \n");  
    printf("Curso: Ciência da Computação");
```

```
}
```



2. Definindo Funções

```
#include <stdio.h>  
#include <stdlib.h>
```

```
void indentifica_aluno(void);
```



Declaração da função.

```
void indentifica_aluno(void) {
```



Definição da função.

```
    printf("Nome: Alexandre Zaghetto \n");  
    printf("Matricula: 123456 \n");  
    printf("Curso: Engenharia de Computação");
```

```
}
```

2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void indentifica_aluno(void);
```



Declaração da função.

```
int main() {
```

```
    indentifica_aluno();
```

```
    return 0;
```

```
}
```

```
void indentifica_aluno(void) {
```



Definição da função.

```
    printf("Nome: Alexandre Zaghetto \n");
```

```
    printf("Matricula: 123456 \n");
```

```
    printf("Curso: Engenharia de Computação");
```

```
}
```

2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void identifica_aluno(void);
```

```
int main() {
    identifica_aluno();
```

Também pode ser chamado de protótipo da função.

```
    return 0;
}
```

Corpo da função.

```
void identifica_aluno(void) {
    printf("Nome: Alexandre Zaghetto \n");
    printf("Matricula: 123456 \n");
    printf("Curso: Engenharia de Computação");
}
```


2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void indentifica_aluno(void);
```

```
int main() {
    indentifica_aluno();

    return 0;
}
```

Programa principal.

```
void indentifica_aluno(void) {
    printf("Nome: Alexandre Zaghetto \n");
    printf("Matricula: 123456 \n");
    printf("Curso: Engenharia de Computação");
}
```

2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void indentifica_aluno(void);
```

```
int main() {
    indentifica_aluno();
    return 0;
}
```

Chamada para a função.

```
void indentifica_aluno(void) {
    printf("Nome: Alexandre Zaghetto \n");
    printf("Matricula: 123456 \n");
    printf("Curso: Engenharia de Computação");
}
```

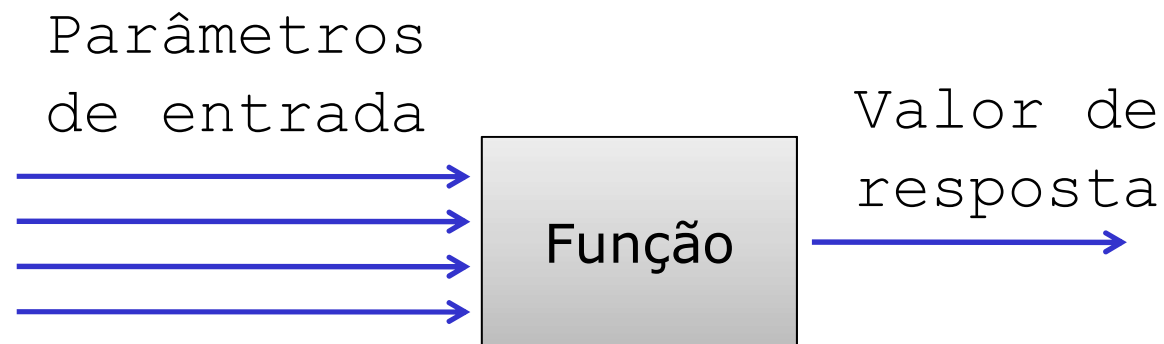
2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.

Exemplo: Escreva uma função que recebe um número inteiro e imprime na tela do computador "PAR!", caso o número seja par, ou "IMPAR!", caso o número seja impar.

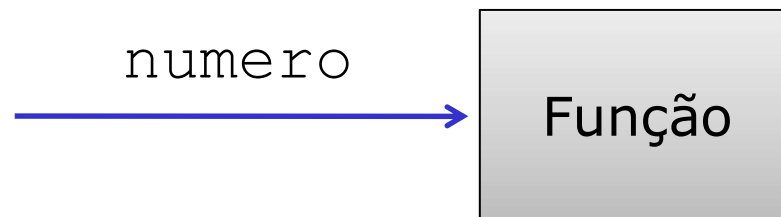
2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.



2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.



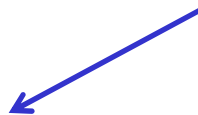


2. Definindo Funções

Não retorna nada.



Recebe um número inteiro.



```
void imparpar(int numero) {
```

```
}
```

2. Definindo Funções

```
void imparpar(int numero) {  
  
    if (numero%2==0)  
        printf("PAR!\n");  
    else  
        printf("IMPAR!\n");  
}
```



2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void imparpar(int);
```

```
void imparpar(int numero) {
    if (numero%2==0)
        printf("PAR!\n");
    else
        printf("IMPAR!\n");
}
```


2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void imparpar(int);
```

→ Declaração da função.

```
int main() {
```

```
    imparpar(5);
```

→ Chamada para a função.

```
    return 0;
```

```
}
```

```
void imparpar(int numero) {
```

→ Definição da função.

```
    if (numero%2==0)
```

```
        printf("PAR!\n");
```

```
    else
```

```
        printf("IMPAR!\n");
```

```
}
```

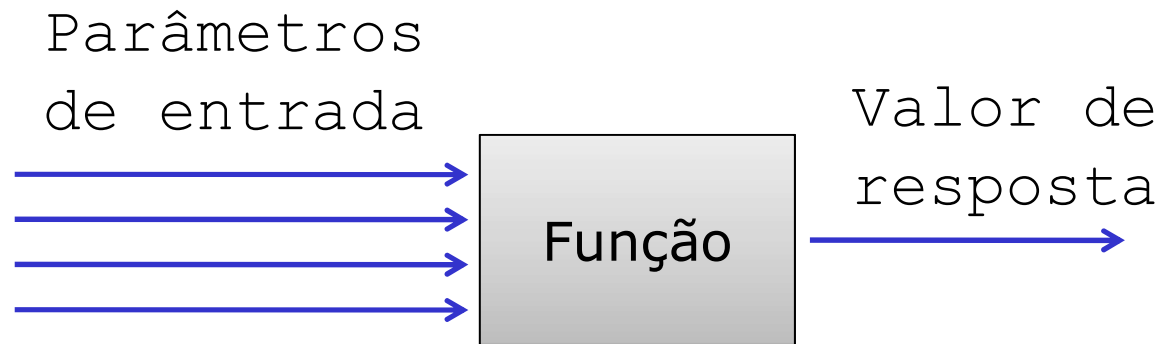
2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.

Exemplo: Escreva uma função que recebe dois números reais e imprime na tela do computador o maior entre eles.

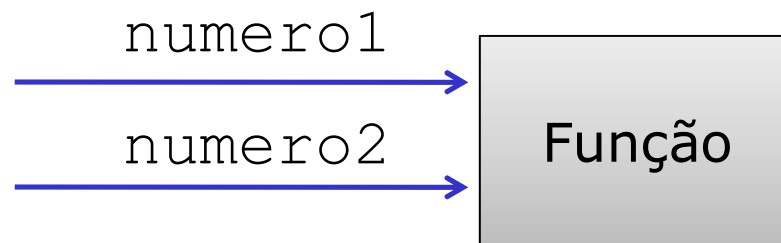
2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.



2. Definindo Funções

- Função que não tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.

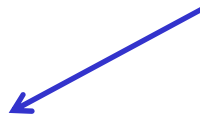


2. Definindo Funções

Não retorna nada.



Recebe dois números reais.



```
void maior (float numero1, float numero2) {
```

```
}
```



2. Definindo Funções

```
void maior (float numero1, float numero2){  
  
    if (numero1>numero2)  
        printf("Maior: %f \n", numero1);  
    else if (numero2>numero1)  
        printf("Maior: %f \n", numero2);  
    else  
        printf("São iguais!");  
}
```



2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
void maior (float, float);
```

```
void maior (float numero1, float numero2){
    if (numero1>numero2)
        printf("Maior: %f \n", numero1);
    else if (numero2>numero1)
        printf("Maior: %f \n", numero2);
    else
        printf("São iguais!");
}
```



2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>

void maior (float, float);

int main() {

    maior(12.3, 12.1);

    return 0;
}

void maior (float numero1, float numero2) {

    if (numero1>numero2)
        printf("Maior: %f \n", numero1);
    else if (numero2>numero1)
        printf("Maior: %f \n", numero2);
    else
        printf("São iguais!");
}
```


6. Gerador de Números Pseudo-aleatórios

Exemplo 2: Um dos mais comuns geradores de números pseudoaleatórios é o *linear congruential generator*, que utiliza a recorência abaixo. A série de valores gerados por este algoritmo é determinada por um número fixo chamado *semente*.

$$X_{n+1} = (aX_n + b) \bmod m$$

Escreva um algoritmo que gere e mostre duas sequencias R1 e R2 de números pseudoaleatórios de comprimento 100000 cada uma. Normaliza as sequencias de forma que os valores fiquem entre 0 e 1. Calcule a média de cada sequência. Considere:

R1 → $X_0 = 5$, $a = 22695477$, $b = 1$ e $m = 1013904223$

R2 → $X_0 = 23$, $a = 22695477$, $b = 1$ e $m = 1013904223$

3. Macros *versus* Funções

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#define N 1000

float geraAleat(int);

int main(){

    int sem; float mean;

    printf("Digite a semente de R1: ");
    scanf("%d", &sem);
    mean = geraAleat(sem);
    printf("Media = %.2f\n", mean);
    printf("Digite a semente de R2: ");
    scanf("%d", &sem);
    mean = geraAleat(sem);
    printf("Media = %.2f\n", mean);

    return 0;
}
```

3. Macros *versus* Funções

```
float geraAleat(int seed){  
  
    float X[N];  
    float numMax, numMin;  
    int i;  
  
    int a = 22695477, b = 1, m = 1013904223;  
  
    X[0] = seed;  
  
    numMin = X[0];  
    for(i = 1; i < N; i++){  
        X[i] = (a*(int)X[i-1]+b)%m;  
  
        if (X[i] < numMin)  
            numMin = X[i];  
    }  
  
    for(i = 0; i < N; i++){  
        X[i] = X[i]-numMin;  
    }  
}
```

3. Macros *versus* Funções

```
numMax = X[0];

for(i = 0; i < N; i++){
    X[i] = X[i]/numMax;
    mean = mean+X[i];
}

mean = mean/N;

for(i = 0; i < N; i++){
    printf("%.2f ", X[i]);
}

return mean;
}
```

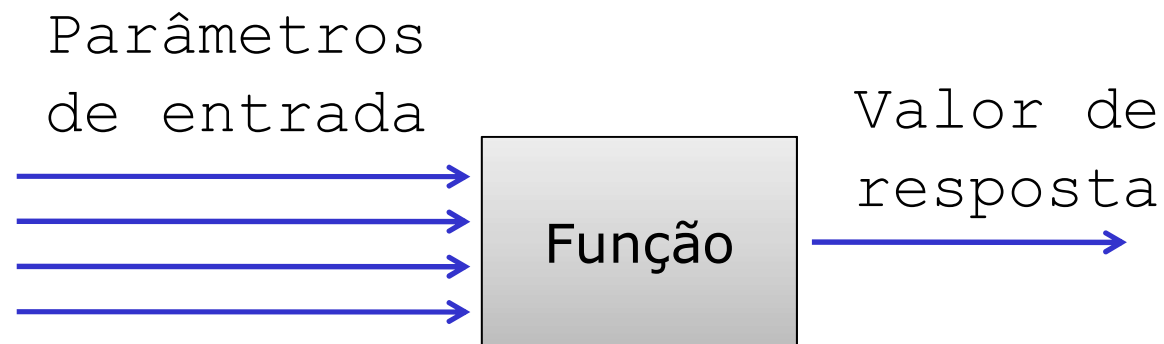
2. Definindo Funções

- Função que tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.

Exemplo: Escreva uma função que recebe um valor inteiro n e um valor real x e retorna um valor real x^n .

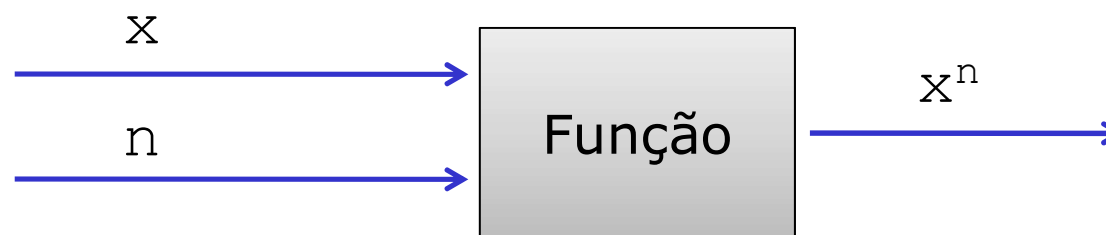
2. Definindo Funções

- Função que tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.



2. Definindo Funções

- Função que tem valor de resposta e que recebe argumentos **por valor** ao ser chamada.





2. Definindo Funções

Retorna um ***float***.



Recebe um ***float*** e um ***int***.



```
float potencia(float x, int n) {  
  
    float pot;  
  
    return pot;  
}
```


2. Definindo Funções

```
float potencia(float x, int n){  
  
    float pot = 1;  
    int i;  
    for(i=0; i<n; i++) pot = pot*x;  
    return pot;  
}
```



2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>
```

```
float potencia(float, int);
```

```
float potencia(float x, int n){

    float pot = 1;
    int i;
    for(i=0; i<n; i++) pot = pot*x;
    return pot;
}
```



2. Definindo Funções

```
#include <stdio.h>
#include <stdlib.h>

float potencia(float, int);

int main() {

    float resultado;

    resultado = potencia(5,2);
    printf("%f \n", resultado);

    return 0;
}

float potencia(float x, int n) {

    float pot = 1;
    int i;
    for(i=0; i<n; i++) pot = pot*x;
    return pot;
}
```



3. Macros *versus* Funções

- Problema com macros:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAIUSC(ch)  ((ch)>= 'a' && (ch)<='z') ? \
                    ((ch) - 'a' + 'A') : (ch)

int main()
{
    char letra_maiusc;

    letra_maiusc = MAIUSC(getchar());

    printf("%c \n\n", letra_maiusc);

    return 0;
}
```



3. Macros *versus* Funções

- Solução com funções:

```
#include <stdio.h>
#include <stdlib.h>

char maiusc(char);

int main() {
    char letra_maiusc;

    letra_maiusc = maiusc(getchar());

    printf("%c \n\n", letra_maiusc);

    return 0;
}

char maiusc(char ch) {
    return (ch >='a' && ch <='z') ? (ch-'a' + 'A') : ch;
}
```



3. Macros *versus* Funções

- Problema com macros:

```
#include <stdio.h>
#include <stdlib.h>
#define MENOR(x,y)      (x<y) ? (x) : (y)

int main() {

    int n1 = 1, n2 = 2, n;

    n = MENOR(n1++, n2++);

    printf("n1=%d \t n2=%d\t n=%d", n1, n2, n);

    return 0;
}
```

3. Macros *versus* Funções

- A solução do problema do *slide* anterior, utilizando funções, fica como exercício.
- Várias funcionalidades em C são implementadas utilizando macros. Para vê-las, examine o arquivo **ctype.h**, que acompanha seu compilador.
- Macros são simples substituições dentro dos programas. Isso torna a execução mais rápida do que se fosse realizada a chamada de funções.
- Por outro lado, o código é aumentado, pois o código da macro será duplicado cada vez que esta for utilizada no programa.
- Uma vantagem (e também desvantagem) do uso de macros é a não necessidade de especificar o tipo dos argumentos.

“No fim das contas, porém, o fato é que *nós educamos a nós mesmos*. Nós aprendemos, antes de tudo, decidindo aprender, assumindo um compromisso com a aprendizagem, que, por sua vez, gera concentração.”

Salman Khan