

# Estruturas de Dados e Algoritmos

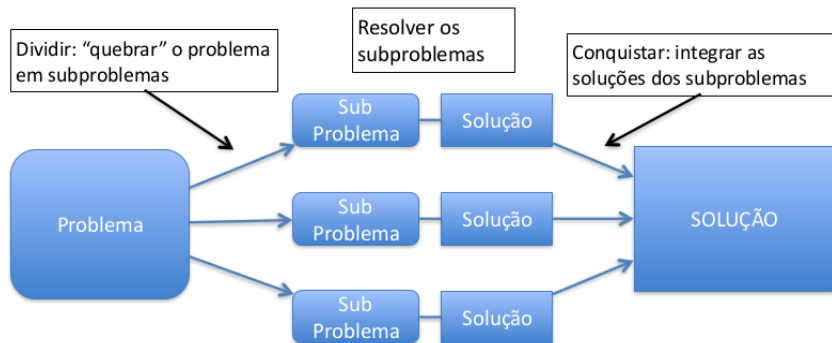
## Funções em C

Professores Anderson e Cesar

IFPB - Campina Grande

# Conceitos Iniciais

- Funções permitem que o mesmo código seja **utilizado várias vezes** e **implementado apenas uma**
- Ao descobrir um erro, só é necessário **corrigir em apenas um local**
- Também facilita a localização de erros

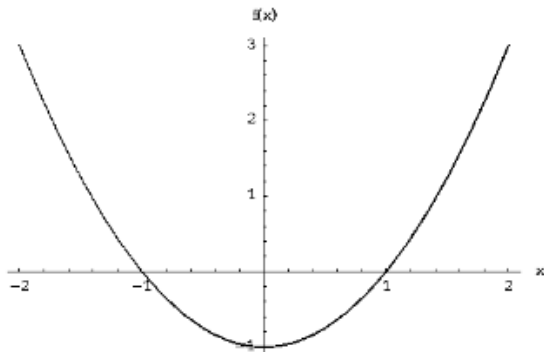


- Um programa em C é composto por um conjunto de funções
  - ▶ São módulos de código que resolvem algum sub-problema
- Nós já usamos um conjunto de funções para escrever nossos programas em C
  - ▶ Ex: *printf()*, *scanf()*, *main()*

# Função Matemática

Nome      Parâmetro      Operações

$$f(x) = x^2 - 1$$



```
int func(int x) {  
    return x*x-1;  
}
```

1

2

3

# Escrevendo Funções em C

Qual a saída do seguinte programa, se for digitado 6?

```
#include <stdio.h>

int func(int x);

int main() {
    int x, y;
    scanf("%d",&x);
    y = func(x);
    printf("%d\n", y);
    return 0;
}

int func(int x) {
    return x*x-1;
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14

```
void escreveResposta(int x) {  
    printf("Resposta: %d\n", x);  
}
```

1

2

3

## Variáveis globais e Variáveis locais

- O escopo de uma variável **define os locais** onde ela pode ser acessada
- Variáveis globais
  - ▶ Podem ser acessadas em **qualquer lugar**
- Variáveis locais
  - ▶ Podem ser acessadas **apenas dentro da função** ou escopo em que foram declaradas
- Em geral deve-se evitar o uso de variáveis globais



## O que ocorre quando uma função é chamada?

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

```
int x = 20, y;
```

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int a = 5;  
    y = func(x);  
    printf("%d %d\n", y, x);  
    return 0;  
}
```

**Parâmetros** e as **variáveis locais** existem apenas dentro da função

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

```
int x = 20, y;
```

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int a = 5;  
    y = func(x);  
    printf("%d %d\n",  
           y, x);  
    return 0;  
}
```

1

2

3

4

5

6

7

8

9

10

11

12

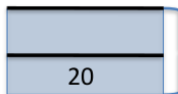
13

14

Memória

y

x



Variáveis  
globais

# Pilha de Execução

**Parâmetros** e as **variáveis locais** existem apenas dentro da função

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

```
int x = 20, y;
```

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int a = 5;  
    y = func(x);  
    printf("%d %d\n",  
           y, x);  
    return 0;  
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

Memória

a

y

x

5

20

Variável local  
do *main*

Variáveis  
globais

**Parâmetros** e as **variáveis locais** existem apenas dentro da função

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

```
int x = 20, y;
```

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int a = 5;  
    y = func(x);  
    printf("%d %d\n",  
           y, x);  
    return 0;  
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

Memória

a

y

x

5

20

Variável local  
do *main*

Variáveis  
globais

**Parâmetros** e as **variáveis locais** existem apenas dentro da função

```
#include <stdio.h>

int x = 20, y;
int func(int x) {
    int b = 10;
    x = x-b;
    return x;
}

int main() {
    int a = 5;
    y = func(x);
    printf("%d %d\n",
           y, x);
    return 0;
}
```

1

2

3

4

5

Memória

6

7

8

9

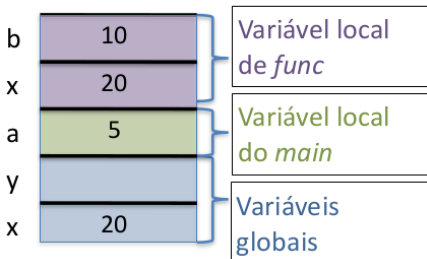
10

11

12

13

14



**Parâmetros** e as **variáveis locais** existem apenas dentro da função

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

```
int x = 20, y;
```

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int a = 5;  
    y = func(x);  
    printf("%d %d\n",  
           y, x);  
    return 0;  
}
```

1

2

3

4

5

6

7

8

9

10

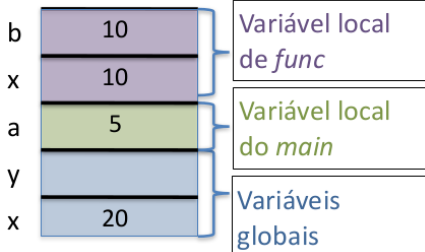
11

12

13

14

Memória



**Parâmetros** e as **variáveis locais** existem apenas dentro da função

```
#include <stdio.h>

int x = 20, y;
int func(int x) {
    int b = 10;
    x = x-b;
    return x;
}

int main() {
    int a = 5;
    y = func(x);
    printf("%d %d\n",
        y, x);
    return 0;
}
```

1

2

3

4

5

6

7

8

9

10

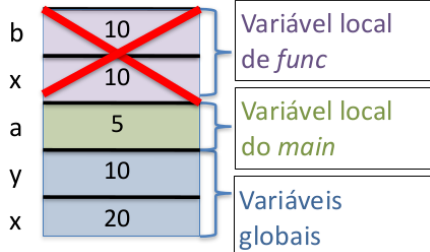
11

12

13

14

Memória



# Pilha de Execução

**Parâmetros** e as **variáveis locais** existem apenas dentro da função

```
#include <stdio.h>

int x = 20, y;
int func(int x) {
    int b = 10;
    x = x-b;
    return x;
}

int main() {
    int a = 5;
    y = func(x);
    printf("%d %d\n",
        y, x);
    return 0;
}
```

1

2

Saída

3

4

10 20

5

6

7

8

Memória

9

10

a

5

Variável local  
do main

11

y

10

12

x

20

Variáveis  
globais

13

14



Qual a saída agora?

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

```
int x = 20, y;
```

```
int func(int a) {
```

```
    int b = 10;
```

```
    x = x-b;
```

```
    return x;
```

```
}
```

```
int main() {
```

```
    int a = 5;
```

```
    y = func(x);
```

```
    printf("%d %d\n", y, x);
```

```
    return 0;
```

```
}
```

## Passagem por Valor e por Referência

- Existem duas formas de passagem de parâmetro
  - ▶ Por valor → como no exemplo anterior
  - ▶ Por referência
    - A variável passada como parâmetro **pode ser modificada** dentro da função
    - Como implementar isso?

## Passagem de Parâmetro por Valor

O valor da variável *a* é **copiado** para a variável local *x*

Mudanças locais em *x* não causam mudanças em *a*

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

```
int func(int x) {  
    int b = 10;  
    x = x-b;  
    return x;  
}
```

```
int main() {  
    int y, a = 5;  
    y = func(a);  
    printf("%d %d\n", y, a);  
    return 0;  
}
```

## Passagem de Parâmetro por Referência

É passado como parâmetro o **endereço** de memória de *a*

Mudanças locais em *\*x* provocam mudanças em *a*

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

```
int func(int *x) {  
    int b = 10;  
    *x = *x - b;  
    return *x;  
}
```

```
int main() {  
    int y, a = 5;  
    y = func(&a);  
    printf("%d %d\n", y, a);  
    return 0;  
}
```

# Arrays como Parâmetro

Essa função funciona para arrays de todos os tamanhos?

```
#include <stdio.h> 1
void modulo(int val[]) { //poderia ser int *val 2
    for(int i = 0; i < 10; i++){ 3
        if(val[i] < 0) 4
            val[i] *= -1; 5
    } 6
} 7
int main() { 8
    int y[10], i; 9
    for(i = 0; i < 10; i++) 10
        y[i] = i*-2; 11
    modulo(y); 12
    for(i = 0; i < 10; i++) 13
        printf("%d ", y[i]); 14
    return 0; 15
} 16
```

# Arrays como Parâmetro – Solução mais genérica

O array sempre é passado como referência, porque?

```
#include <stdio.h>
void modulo(int val[], int n) {
    for(int i = 0; i < n; i++){
        if(val[i] < 0)
            val[i] *= -1;
    }
}
int main() {
    int y[10], i;
    for(i = 0; i < 10; i++)
        y[i] = i*-2;
    modulo(y, 10);
    for(i = 0; i < 10; i++)
        printf("%d ", y[i]);
    return 0;
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

# Matrizes como Parâmetro


Essa função funciona para arrays de tamanhos variados

Deve-se passar como parâmetro antes a quantidade de linhas e de colunas

```
#include <stdio.h>
int soma_mat(int l, int c, int mat[l][c]) {
    int s = 0, i, j;
    for(i = 0; i < l; i++) {
        for(j = 0; j < c; j++) {
            s += mat[i][j];
        }
    }
    return s;
}

int main() {
    int mat[3][4];
    int i, j;
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 4; j++) {
            mat[i][j] = i+j;
        }
    }
    printf("%d\n", soma_mat(3,4,mat));
    return 0;
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21



Slides Adaptados do Professor Ruan.