

Estrutura de Dados: Funções

Prof. Jean Nunes

Função

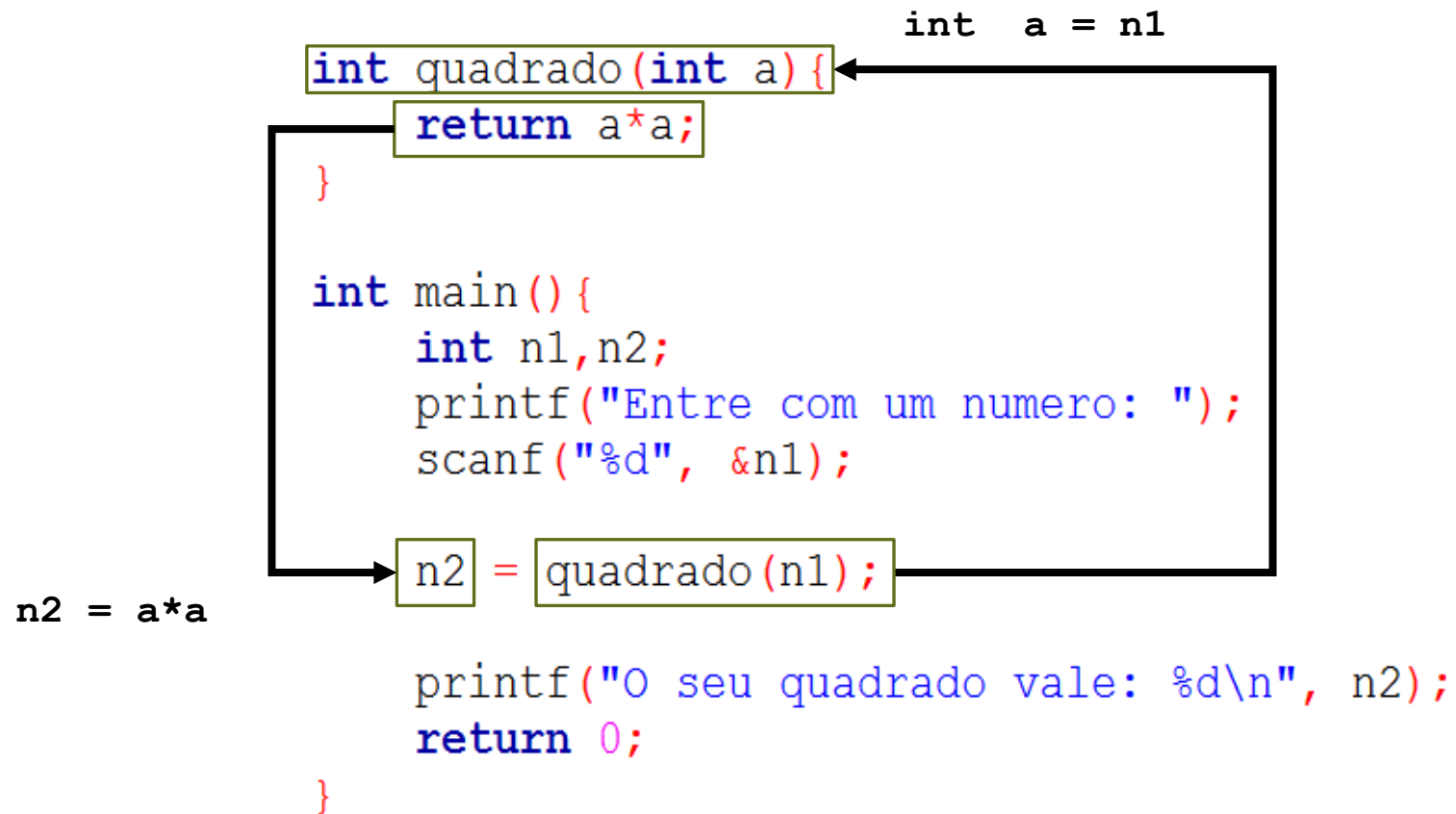
- Blocos de código nomeados e chamados de dentro de um programa.
 - **printf()**: função que escreve na tela
 - **scanf()**: função que lê o teclado

Função

- **Estruturação:** programas grandes e complexos são construídos bloco a bloco.
- **Reutilização:** evita a cópia desnecessária de trechos de código que realizam a mesma tarefa, diminuindo assim o tamanho do programa e a ocorrência de erros.

Ordem de Execução

O **programa** que chama a função é pausado até que ela termine a sua execução.



Estrutura

Valor que a
função pode ou
não retornar

Nome que
identifica a função

Lista de valores
passados para a
função processar

```
tipo_retornado nome_função (parâmetros) {  
    conjunto de declarações e comandos  
}
```

Corpo da função (tarefas
que ela executa)

Corpo

- Não é recomendado leitura/escrita dentro da função.
 - Função é para realizar tarefa específica e bem-definida.
 - As operações de entrada/saída (ex. **scanf()** e **printf()**) devem ser feitas em quem chamou a função (ex., na **main()**).
 - Garante generalidade.

Parâmetros

lista de variáveis
juntamente com seus
tipos:

- *tipo1 nome1, tipo2 nome2, ... ,
tipoN nomeN*



```
//Declaração CORRETA de parâmetros
int soma(int x, int y) {
    return x + y;
}

//Declaração ERRADA de parâmetros
int soma(int x, y) {
    return x + y;
}
```


Parâmetros

É por meio dos parâmetros que a função recebe informação do programa que a chamou.

```
int x = 2;  
int y = 3;
```

```
int soma(int x, int y) {  
    return x + y;  
}  
  
int main() {  
    int z = soma(2, 3);  
  
    return 0;  
}
```

```
int soma(int x, int y) {  
  
    scanf("%d", &x);  
    scanf("%d", &y);  
  
    return x + y;  
}
```



Parâmetros

- Uma função pode não ter parâmetro de entrada.

```
void imprime() {  
    printf("Teste\n");  
}
```

```
void imprime(void) {  
    printf("Teste\n");  
}
```

Retorno

- Uma função pode ou não retornar um valor
 - Se ela retornar um valor, alguém deverá receber este valor
- Podemos retornar qualquer valor válido em C
 - tipos pré-definidos: int, char, float e double
 - tipos definidos pelo usuário: struct

Comando return

Forma geral:

- **return** *valor ou expressão*;
- **return**; (usada para terminar uma função que não retorna valor)

O valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Comando return

Função com retorno de valor

```
int soma(int x, int y){  
    return x + y;  
}  
  
int main(){  
    int z = soma(2,3);  
  
    return 0;  
}
```

Função sem retorno de valor

```
void imprime(){  
    printf("Teste\n");  
}  
  
int main(){  
    imprime();  
  
    return 0;  
}
```

Comando return

- Uma função pode ter mais de uma declaração **return**.
- Quando o comando **return** é executado, a função termina imediatamente e os comandos restantes são **ignorados**.

```
int maior(int x, int y){  
    if(x > y)  
        return x;  
    else  
        return y;  
    printf("Esse texto nao sera impresso\n");  
}
```

Declaração de Funções

- Devem ser declaradas antes de serem utilizadas.

```
int quadrado(int a) {  
    return a*a;  
}
```

```
int main() {  
    int n1, n2;  
    printf("Entre com um numero: ");  
    scanf("%d", &n1);  
  
    n2 = quadrado(n1);  
  
    printf("O seu quadrado vale: %d\n", n2);  
    return 0;  
}
```


Declaração de Funções

Pode-se definir apenas o protótipo da função antes da cláusula **main**.

Posso adicionar o protótipo em arquivo separado?

```
int quadrado(int a);
```

```
int main() {  
    int n1, n2;  
    printf("Entre com um numero: ");  
    scanf("%d", &n1);  
  
    n2 = quadrado(n1);  
  
    printf("O seu quadrado vale: %d\n", n2);  
    return 0;  
}  
  
int quadrado(int a) {  
    return a*a;  
}
```

Escopo

- O escopo é o conjunto de regras que determina o uso e a validade de variáveis nas diversas partes do programa.
 - Variáveis Locais
 - Variáveis Globais
 - Parâmetros formais

Escopo

```
int fatorial (int n) {  
    if (n == 0)  
        return 1;  
    else {  
        int i;  
        int f = 1;  
        for (i = 1; i <= n; i++)  
            f = f * i;  
        return f;  
    }  
}
```

Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas.

Escopo

- Parâmetros formais são declarados como sendo as entradas de uma função.
- É uma variável local da função.

```
float quadrado(float x);
```

Escopo

Variáveis globais são declaradas fora de todas as funções do programa.

Evite variáveis globais!

Elas são conhecidas e podem ser alteradas por todas as funções do programa.

Uma variável local com o mesmo nome de uma variável global, a função dará preferência à variável local.

Passagem de Parâmetros

Quando parâmetros de uma função são passados por *valor*, **uma cópia do valor do parâmetro é feita e passada para a função.**

Mesmo que esse valor mude dentro da função, nada acontece com o valor de fora da função.

Passagem por valor

```
void incrementa(int n) {  
    n = n + 1;  
  
    printf("Dentro da funcao: x = %d\n", n);  
}  
  
int main() {  
    int x = 5;  
    printf("Antes da funcao: x = %d\n", x);  
  
    incrementa(x);  
  
    printf("Depois da funcao: x = %d\n", x);  
    return 0;  
}
```

Saída:

Antes da funcao: x = 5

Dentro da funcao: x = 6

Depois da funcao: x = 5

Passagem por referência

Quando se quer que o valor da variável mude dentro da função, usa-se passagem de parâmetro por *referência*.

Neste tipo de chamada, não se passa para a função o valor da variável, mas a sua *referência* (seu endereço na memória);

Passagem por referência

Ex: função **scanf()**

```
int main() {  
    int x = 5;  
    printf("Antes do scanf: x = %d\n", x);  
    printf("Digite um numero: ");  
    scanf("%d", &x);  
    printf("Depois do scanf: x = %d\n", x);  
  
    return 0;  
}
```


Passagem por referência

Coloca-se um asterisco “*” na frente do nome do parâmetro na declaração da função:

```
//passagem de parâmetro por valor  
void incrementa(int n);
```

```
//passagem de parâmetro por referência  
void incrementa(int *n);
```

Ao se chamar a função, é necessário agora utilizar o operador “&”.

```
//passagem de parâmetro por valor  
int x = 10;  
incrementa(x);
```

```
//passagem de parâmetro por referência  
int x = 10;  
incrementa(&x);
```

Passagem por referência

```
//passagem de parâmetro por valor
void incrementa(int n){
    n = n + 1;
}
//passagem de parâmetro por referência
void incrementa(int *n){
    *n = *n + 1;
}
```

No corpo da função, é necessário colocar um asterisco “*” sempre que se desejar acessar o conteúdo do parâmetro passado por referência.

Passagem por referência

```
void incrementa(int *n) {  
    *n = *n + 1;  
  
    printf("Dentro da funcao: x = %d\n", n);  
}
```

```
int main() {  
    int x = 5;  
    printf("Antes da funcao: x = %d\n", x);
```

```
    incrementa(&x);
```

```
    printf("Depois da funcao: x = %d\n", x);  
    return 0;
```

```
}
```

`int *n = &x;`

Saída:

Antes da funcao: x = 5

Dentro da funcao: x = 6

Depois da funcao: x = 6



Exercício

- Crie uma função que troque o valor de dois números inteiros passados por referência.

```
void Troca (int*a, int*b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

Exercício

Crie uma função que troque o valor de dois números inteiros passados por referência.

Arrays como parâmetro

Arrays são sempre passados por referência para uma função;

- Evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa.

Arrays como parâmetro

- É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar para a função o tamanho do array separadamente.
- Quando passamos um array por parâmetro, independente do seu tipo, o que é de fato passado é o endereço do primeiro elemento do array.

Arrays como parâmetro

- Na passagem de um array como parâmetro de uma função podemos declarar a função de diferentes maneiras, todas equivalentes:

```
void imprime(int *m, int n);  
void imprime(int m[], int n);  
void imprime(int m[5], int n);
```

Arrays como parâmetro

Exemplo: Função que imprime um array

```
void imprime(int *m, int n) {  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}  
  
int main () {  
    int vet[5] = {1, 2, 3, 4, 5};  
    imprime(vet, 5);  
  
    return 0;  
}
```

Memória		
posição	variável	conteúdo
119		
120		
121	int m	123
122	int n	5
123	vet[0]	1
124	vet[1]	2
125	vet[2]	3
126	vet[3]	4
127	vet[4]	5
128		

Struct como parâmetro

- Cada campo da struct é como uma variável independente. Ela pode, portanto, ser passada individualmente por *valor* ou por *referência*

Struct como parâmetro

- Passar por parâmetro toda a struct
 - **Passagem por valor**
 - A struct é tratada com uma variável qualquer e seu valor é copiado para dentro da função

Struct como parâmetro

- **Passagem por referência**
 - Valem as regras de uso do asterisco “*” e operador de endereço “&”
 - Devemos acessar o conteúdo da struct para somente depois acessar os seus campos e modificá-los.
 - Uma alternativa é usar o *operador seta* “->”

Struct como parâmetro

Usando “*”

```
struct ponto {  
    int x, y;  
};  
  
void atribui(struct ponto *p) {  
    (*p).x = 10;  
    (*p).y = 20;  
}  
  
struct ponto p1;  
  
atribui(&p1);
```

Usando “->”

```
struct ponto {  
    int x, y;  
};  
  
void atribui(struct ponto *p) {  
    p->x = 10;  
    p->y = 20;  
}  
  
struct ponto p1;  
  
atribui(&p1);
```