



UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO DE SISTEMAS DE INFORMAÇÃO
PICOS - PI

REVISÃO DA LINGUAGEM C

Prof. Ma. Luana Batista da Cruz
luana.b.cruz@nca.ufma.br

Agosto - 2021

Roteiro

2

- Introdução a Linguagem C
- Vetores e Matrizes
- Registros
- Funções

Introdução a Linguagem C

Tipos básicos de variáveis

Operadores aritméticos, relacionais e lógicos

Comando de saída e entrada

Comandos de repetição

Introdução a Linguagem C

4

- ❑ A linguagem C pertence a uma família de linguagens cujas características são:
 - Modularidade
 - Compilação separada
 - Recursos de baixo nível
 - Confiabilidade
 - Simplicidade
 - Facilidade de uso

Introdução a Linguagem C

5

❑ Tipos Básicos de Variáveis

- ❑ **char:** o valor armazenado é um caractere. Caracteres geralmente são armazenados em códigos (usualmente o código ASCII)
- ❑ **int:** número inteiro
- ❑ **float:** número em ponto flutuante de precisão simples. São conhecidos normalmente como números reais
- ❑ **double:** número em ponto flutuante de precisão dupla
- ❑ **void:** este tipo serve para indicar que um resultado não tem um tipo definido

Introdução a Linguagem C

6

❑ Modificadores de Tipos

- Podem aumentar ou diminuir a capacidade de armazenamento e definir se a faixa numérica será a positiva ou então negativa
 - **signed**: números positivos e negativos
 - **unsigned**: números positivos
 - **long**: aumentar a capacidade de armazenamento
 - **short**: diminuir a capacidade de armazenamento

Introdução a Linguagem C

7

❏ Modificadores de Tipos

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
long int	4	-4.294.967.295 a 4.294.967.295
unsigned long int	4	0 a 4.294.967.295
float	4	Seis dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

Introdução a Linguagem C

8

❑ Variáveis

- ❑ Regras básicas para nomear variáveis:
 - Todo nome só pode conter letras e/ou dígitos
 - Apenas o caractere símbolo "_" pode ser usado
 - Todo primeiro caractere deve ser sempre uma letra
 - Letras maiúsculas e minúsculas são consideradas caracteres diferentes
- ❑ Declaração de variáveis:
 - `int i, idade, numero;`
 - `float salario, altura;`
 - `unsigned char sexo, letra;`
- ❑ Atribuição:
 - `idade = 31;`
 - `sexo = 'm';`

Introdução a Linguagem C

9

❑ Variáveis

❑ Regras básicas para nomear variáveis:

- Todo nome só pode conter letras e/ou dígitos
- Apenas o caractere símbolo "_" pode ser usado
- Todo primeiro caractere deve ser sempre uma letra
- Letras maiúsculas e minúsculas são consideradas caracteres diferentes

❑ Declaração de variáveis:

- `int i, idade, numero;`
- `float salario, altura;`
- `unsigned char sexo, letra;`

❑ Atribuição:

- `idade = 31;`
- `sexo = 'm';`

Obs.: não só as variáveis mas toda a linguagem C é “**Case Sensitive**”, isto é, maiúsculas e minúsculas fazem diferença. Por exemplo: **Idade** \neq **idade**, ou seja, são duas variáveis diferentes.

Introdução a Linguagem C

10

❑ Variáveis

❑ Regras básicas para nomear variáveis:

- Todo nome só pode conter letras e/ou dígitos
- Apenas o caractere símbolo "_" pode ser usado
- Todo primeiro caractere deve ser sempre uma letra
- Letras maiúsculas e minúsculas são consideradas caracteres diferentes

❑ Declaração de variáveis:

- `int i, idade, numero;`
- `float salario, altura;`
- `unsigned char sexo, letra;`

❑ Atribuição:

- `idade = 31;`
- `sexo = 'm';`

Caracteres usam
aspas simples

Obs.: não só as variáveis mas toda a linguagem C é “**Case Sensitive**”, isto é, maiúsculas e minúsculas fazem diferença. Por exemplo: **Idade** \neq **idade**, ou seja, são duas variáveis diferentes.

Introdução a Linguagem C

11

❑ Operadores Aritméticos Básicos

Operador	Símbolo	Exemplo
Adição	+	$a + b$
Subtração	-	$a - b$
Multiplicação	*	$a * b$
Divisão	/	a / b
Resto de Divisão Inteira	%	$a \% b$

Introdução a Linguagem C

12

❑ Operadores Relacionais e Lógicos

Operador	Símbolo	Exemplo
Igual	<code>==</code>	<code>a == b</code>
Diferente	<code>!=</code>	<code>a != b</code>
Maior	<code>></code>	<code>a > b</code>
Maior ou igual	<code>>=</code>	<code>a ≥ b</code>
Menor	<code><</code>	<code>a < b</code>
Menor ou igual	<code><=</code>	<code>a ≤ b</code>
Conjunção (e)	<code>&&</code>	<code>a && b</code>
Disjunção (ou)	<code> </code>	<code>a b</code>
Negação	<code>!</code>	<code>! c</code>

Introdução a Linguagem C

13

❑ Operadores Relacionais e Lógicos

Operador	Símbolo	Exemplo	Relacionais
Igual	==	a == b	
Diferente	!=	a != b	
Maior	>	a > b	
Maior ou igual	>=	a ≥ b	
Menor	<	a < b	
Menor ou igual	<=	a ≤ b	Lógicos
Conjunção (e)	&&	a && b	
Disjunção (ou)		a b	
Negação	!	! c	

Introdução a Linguagem C

14

❑ Operações de Fluxo

❑ Comando se

```
if (<condição>)  
    <comandos>;  
[ else  
    <comandos>; ]
```

❑ Comando enquanto

```
while (<condição>)  
    <comandos>;
```

Introdução a Linguagem C

15

❑ Comando de Saída *printf*

- ❑ `printf (<info. de controle>, <lista de variáveis>);`
- ❑ Informações de controle:
 - É uma descrição do que vai aparecer na tela. Também é a definição do tipo de dado do valor a ser exibido (geralmente de uma variável). Isto é feito usando-se os códigos de controle, que usam a notação %

Código	Significado
%d	Inteiro
%f	Float
%c	Caractere
%s	String
%%	Coloca na tela um %

Introdução a Linguagem C

16

❑ Comando de Saída *printf*

❑ Exemplos:

- `printf ("%f", 40.345)`
 - "40.345"
- `printf ("Um caractere %c e um inteiro %d", 'D', 120)`
 - "Um caractere D e um inteiro 120"
- `printf ("%s eh um exemplo", "Este")`
 - "Este eh um exemplo"
- `printf ("%s%d%%", "Juros de ", 10)`
 - "Juros de 10%"

Introdução a Linguagem C

17

❑ Comando de Entrada *scanf*

- ❑ `scanf (<info. de controle>, &<lista de variáveis>);`
- ❑ Exemplos:
 - `scanf ("%f", &salario);`
 - `scanf ("%d", &idade);`
 - `scanf ("%c", &letra);`
 - `scanf ("%d %f %c", &idade, &salario, &letra);`
- ❑ O caractere **&** indica que o valor será armazenado no endereço de memória da variável

Introdução a Linguagem C

18

❑ Caracteres de Escape

Caractere	Significado
<code>\a</code>	aviso sonoro
<code>\n</code>	nova linha
<code>\t</code>	tabulação horizontal
<code>\v</code>	tabulação vertical
<code>\\</code>	caractere de barra invertida
<code>\'</code>	apóstrofe
<code>\"</code>	aspas
<code>\?</code>	interrogação

Exemplo (Estrutura Básica)

19

#include inclui a biblioteca **stdio.h**. Essa biblioteca possui declarações de funções de I/O.

int indica que a função **main** retornará um valor do tipo inteiro.

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

```
int main () {  
    printf ("Alô mundo!\n");  
    return 0;  
}
```

Os caracteres chave **{** e **}** delimitam o início e fim da função **main**, respectivamente.

A função **main** será a primeira a ser chamada quando o programa for executado.

return retorna um valor da função **main**.

Exemplo

20

- ❑ Inserindo `system("PAUSE")` para fazer o programa “parar”:

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

int main(){
    printf ("Alô mundo!\n");
    system ("PAUSE");
    return 0;
}
```

O arquivo **stdlib.h** possui funções de alocação de memória, controle de processos, conversões e outras.

“Para” a execução do programa.

Exemplo

21

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

int main(){
    printf ("Teste %% %%\n");
    printf ("%f\n",40.345);
    printf ("Um caractere %c e um inteiro %d\n",'D',120);
    printf ("%s eh um exemplo\n","Este");
    printf ("%s%d%%\n","Juros de ",10);

    system ("PAUSE");
    return 0;
}
```

Exemplo

22

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

int main(){
    int idade;
    printf ("Digite um número:");
    scanf ("%d", &idade);
    if (idade >= 18)
        printf ("de maior\n");
    else
        printf ("de menor\n");

    system ("PAUSE");
    return 0;
}
```

Exemplo

23

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

int main(){
    int i;
    i = 1;
    while (i <= 20) {
        printf ("Numero = %d\n", i);
        i = i + 1;
    }

    system ("PAUSE");
    return 0;
}
```

Introdução a Linguagem C

24

❑ Variáveis Booleanas

- A linguagem C não possui explicitamente variáveis do tipo booleano. Entretanto, a linguagem considera um número com valor 0 (zero) igual a falso e qualquer número **diferente de 0 (zero) igual a verdadeiro**

Introdução a Linguagem C

25

❑ Caracteres

- A linguagem C trata os caracteres como sendo uma variáveis de um byte (8 bits). Um char também pode ser usado para armazenar números

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

int main(){
    char Ch;
    Ch='A';
    printf ("Caractere = %c\n",Ch);
    printf ("ASCII = %d\n",Ch);

    system ("PAUSE");
    return 0;
}
```

Introdução a Linguagem C

26

❑ Caracteres

- Algoritmo para listar a tabela ASCII

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

int main() {
    unsigned char simbolo = 0;
    while (simbolo < 255){
        printf ("%c=%d  ", simbolo, simbolo);
        simbolo = simbolo + 1;
    }
    system ("PAUSE");
    return 0;
}
```

Introdução a Linguagem C

27

❑ Caracteres

- Funções de entrada para caracteres
 - getch(): apenas retorna o caractere pressionado sem mostrá-lo na tela
 - getche(): mostra o caractere na tela antes de retorná-lo

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

int main(){
    char Ch;
    Ch = getch();
    printf ("Tecla = %c\n",Ch);

    system ("PAUSE");
    return 0;
}
```

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

int main(){
    char Ch;
    Ch = getche();
    printf ("\nTecla = %c\n",Ch);

    system ("PAUSE");
    return 0;
}
```

Introdução a Linguagem C

28

□ String

- Uma string é um vetor (cadeia) de caracteres
- Em C, uma string é terminada com o caractere nulo (código igual a 0 em ASCII) e ele pode ser escrito como `'\0'`

Introdução a Linguagem C

29

□ String

- Uma string é um vetor (cadeia) de caracteres
- Em C, uma string é terminada com o caractere nulo (código igual a 0 em ASCII) e ele pode ser escrito como `'\0'`

string.h é a biblioteca que contém funções de manipulação de strings.

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

Strings podem ser atribuídas diretamente na sua declaração.

```
int main (){
    char frase[100] = "Estrutura de Dados I";
    printf ("Frase = %s\n", frase);
    strcpy (frase, "Estrutura de Dados II");
    printf ("Frase = %s\n", frase);

    system ("PAUSE");
    return 0;
}
```

strcpy é uma função que copia uma string para uma variável

Strings usam aspas duplas.

Introdução a Linguagem C

30

□ String

- gets(): lê uma string do teclado

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

int main (){
    char string[100];
    printf ("Digite o seu nome: ");
    gets (string);
    printf ("\n\n Ola %s",string);

    system ("PAUSE");
    return 0;
}
```

Introdução a Linguagem C

31

□ Abreviação de Expressões

- A linguagem C admite as seguintes equivalências, que podem ser usadas para simplificar expressões ou para facilitar o entendimento de um programa

Expressão Original	Expressão Equivalente
$x = x + k;$	$x += k;$
$x = x - k;$	$x -= k;$
$x = x * k;$	$x *= k;$
$x = x / k;$	$x /= k;$
$x = x + 1$	$x ++$
	$++ x$
$x = x - 1$	$x --$
	$-- x$

Introdução a Linguagem C

32

- ❑ **Comando de Repetição do *while***
 - O comando ***do while*** permite que um certo trecho de programa seja executado **ENQUANTO** uma certa condição for verdadeira

Introdução a Linguagem C

33

❑ Comando de Repetição do *while*

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

int main (){
    int i = 1;
    do {
        printf ("Numero = %d\n", i);
        i = i + 1;
    } while(i <=4);

    system ("PAUSE");
    return (0);
}
```

A utilização dos caracteres de chaves { (início) e } (fim) são obrigatórios no comando **do** **while**.

Introdução a Linguagem C

34

- **Comando de Repetição *for***
 - O comando ***for*** permite que um certo trecho de programa seja executado um **determinado número de vezes**

Introdução a Linguagem C

35

❑ Comando de Repetição *for*

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

int main (){
    int i = 1;
    for (i = 1; i <= 4; i++)
        printf ("Numero = %d\n", i);

    system ("PAUSE");
    return (0);
}
```

Se o **for** tiver mais de um comando é necessário a utilização dos caracteres de chaves para marcar o início e fim do comando: { (início) e } (fim).

Introdução a Linguagem C

36

- ❑ **Comando de Seleção *switch***
 - O conteúdo de uma variável é **comparado** com um valor constante, e caso a comparação seja verdadeira, **um determinado comando é executado**

Introdução a Linguagem C

37

❑ Comando de Seleção *switch*

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    switch (num) {
        case 9: printf ("\n\nO numero eh igual a 9.\n");
                break;
        case 10: printf ("\n\nO numero eh igual a 10.\n");
                break;
        case 11: printf ("\n\nO numero eh igual a 11.\n");
                break;
        default: printf ("\n\nO numero nao eh nem 9 nem 10 nem 11.\n");
    }
    system ("PAUSE");
    return (0);
}
```

Introdução a Linguagem C

38

❑ **Comando *break***

- Pode quebrar a execução de um comando (como no caso do switch) ou interromper a execução de qualquer loop. O break faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido

Introdução a Linguagem C

39

Comentários

Tipos de comentários:

- Comentário de uma linha: //
- Comentário de múltiplas linhas: /* */

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

int main (){
    //Comentário de uma linha
    printf ("parou ...\n");
    /*
    Comentário de múltiplas
    linhas.
    */
    system ("PAUSE");
    return (0);
}
```

40

Vetores e Matrizes

Estruturas de dados homogêneas

Estruturas de Dados Homogêneas

41

- ❑ As **estruturas homogêneas** são conjuntos de dados formados pelo mesmo tipo de dado primitivo. Elas permitem agrupar diversas informações dentro de uma mesma variável
- ❑ Este agrupamento ocorrerá obedecendo sempre ao **mesmo tipo de dado**, e é por esta razão que estas estruturas são chamadas homogêneas
- ❑ A utilização deste tipo de estrutura de dados recebe diversos nomes, como: variáveis indexadas, variáveis compostas, arranjos, vetores, matrizes, tabelas em memória ou *arrays*
- ❑ Os nomes mais usados e que utilizaremos para estruturas homogêneas são: **matrizes** e **vetores** (matriz de uma linha e várias colunas)

Vetores

42

- ❑ É um arranjo de elementos armazenados em memória principal, um após o outro, todos utilizando o mesmo nome de variável

Índices	0	1	2	3	4	5	6	7
Vetor A	314			0	4	16		-3

- ❑ Um valor de um vetor pode ser acessado a partir de seu índice. Ex: `A[0] = 314; printf("%d", A[0]);`
- ❑ Um vetor sempre começa com o índice de valor igual a zero e termina com o valor de seu tamanho menos um

Vetores

Exemplo

43

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

```
int v[3];
```

Declaração de uma variável vetor: o número 3 (três) indica que o vetor terá três elementos.

```
int main(){
```

```
    v[0] = 4;
```

```
    v[1] = 5;
```

```
    v[2] = 6;
```

Atribuição em um elemento de um vetor: o vetor sempre começa com índice 0 (zero).

```
    printf ("V[0]= %d V[1]= %d V[2]= %d \n", v[0], v[1], v[2]);
```

```
    printf ("Somatório %d \n", v[0] + v[1] + v[2]);
```

```
    system ("PAUSE");
```

```
    return 0;
```

```
}
```

Vetores

Exemplo

44

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

int i, soma, v[8] = {1,23,17,4,-5,100, 4, 0};

int main(){
    for(i=0; i<8; i++)
        printf ("V[%d]= %d \n", i, v[i]);

    soma = 0;
    for(i=0; i<8; i++)
        soma += v[i];

    printf ("Soma = %d \n", soma);

    system ("PAUSE");
    return 0;
}
```

Vetores

Exemplo

45

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

int i, soma = 0, idades[30];
float media;

int main(){
    for(i=0; i<30; i++){
        printf ("Digite a %da. idade: \n", i+1);
        scanf ("%d", &idades[i]);
    }
    for(i=0; i<30; i++)
        soma += idades[i];
    media = soma/30;
    printf ("Soma = %d \n", soma);
    printf ("Media = %f \n", media);
    system ("PAUSE");
    return 0;
}
```

Vetores

Exemplo: busca sequencial

46

```
#include <stdio.h>
#include <stdlib.h>
int i, num, achou, v[8] = {1,23,17,4,5,100, 6, 0};
int main(){
    printf ("Digite um número:\n");
    scanf ("%d", &num);
    achou = 0;
    for(i=0; i<8; i++)
        if(v[i] == num){
            achou = 1;
            break;
        }
    if(achou)
        printf ("Numero %d encontrado!\n", num);
    else
        printf ("Numero %d nao encontrado!\n", num);
    system ("PAUSE");
    return 0;
}
```

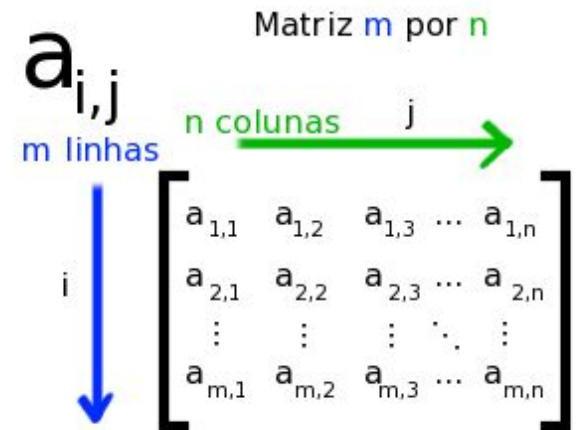
Matrizes

47

- Uma matriz é um vetor com mais de uma dimensão

Índices

	0	1	2	3
0				
1			17	
2	314			
3				
4				



- Um valor de uma matriz pode ser acessado a partir de seus índices. Ex: `M[2][0] = 314; printf("%d", A[1][2]);`
- Um vetor sempre começa com o índice de valor igual a zero e termina com o valor de seu tamanho menos um

Matrizes

Exemplo

48

```
#include <stdio.h>
#include <stdlib.h>
int i, j, m[2][3];

int main(){
    for(i=0; i<2; i++)
        for(j=0; j<3; j++)
            scanf("%d", &m[i][j]);

    for(i=0; i<2; i++){
        printf ("\n");
        for(j=0; j<3; j++)
            printf ("M[%d][%d]= %d, ", j, i, m[i][j]);
    }

    system ("PAUSE");
    return 0;
}
```


Estruturas de dados heterogêneas

Estruturas de Dados Heterogêneas

50

- ❑ As **estruturas heterogêneas** permitem a manipulação de um conjunto de informações de tipos de dados primitivos diferentes, mas que possuem um relacionamento lógico entre si
- ❑ A estrutura heterogênea mais usada e que utilizaremos: registro ou struct

Registros

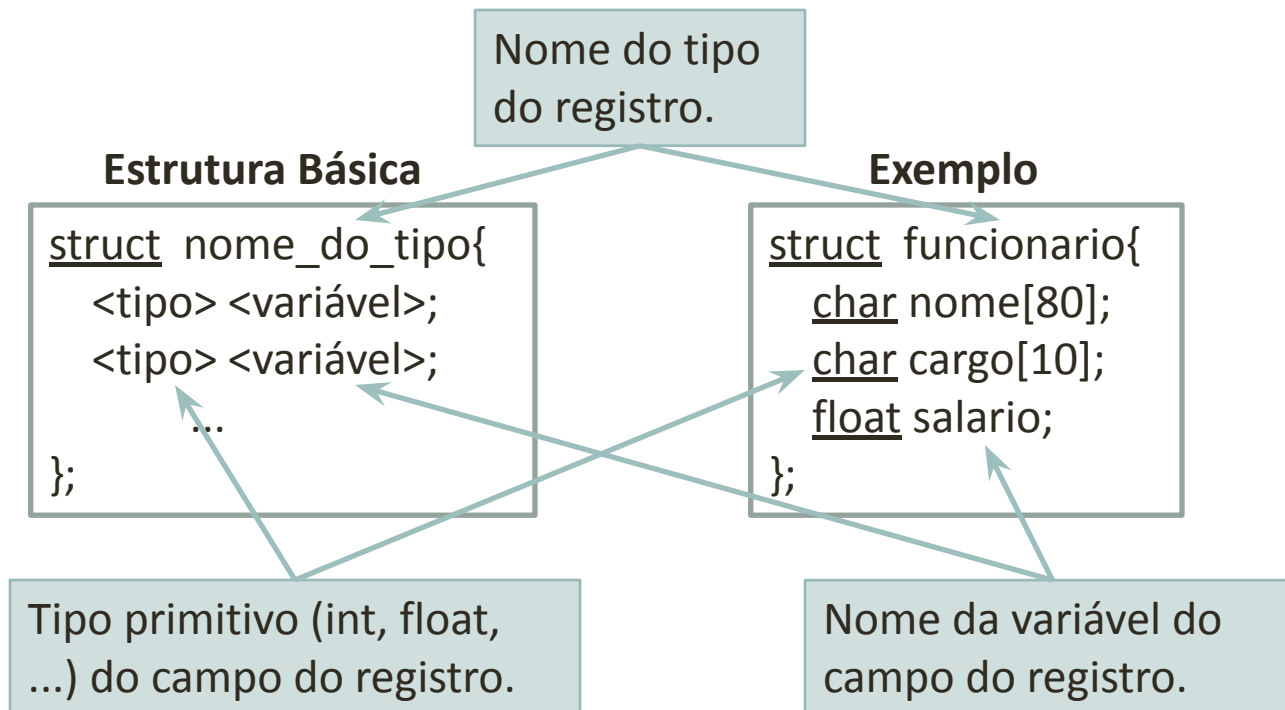
51

- ❑ Um **registro** é uma coleção de várias variáveis, possivelmente de tipos diferentes, e logicamente relacionadas
- ❑ Os elementos de um registro são chamados de campos
- ❑ Exemplos:
 - Funcionário de uma empresa
 - Nome, cargo, salário
 - Aluno universitário
 - Matrícula, nome, curso
 - Endereço
 - CEP, logradouro, numero, bairro, cidade

Declaração de Tipos de Registros

52

- Na linguagem C, registros são conhecidos como structs (abreviatura de structures). A declaração de um registro é a declaração de um tipo personalizado que será utilizado por uma variável



Declaração de Tipos de Registros

53

- ❑ Após o tipo de um registro ser declarado, devemos declarar variáveis para o tipo criado

```
struct funcionario{  
    char nome[80];  
    char cargo[10];  
    float salario;  
};  
struct funcionario func01, func02;
```

- ❑ O acesso ao campo de um registro é feito a utilizando a sintaxe: <nome_da_variável>.<nome_do_campo>

```
...  
func02.nome = "Luana Batista"  
func02.cargo = "professora";  
func02.salario = 2000,00;  
...
```

Registros

Exemplo

54

```
#include <stdio.h>
#include <string.h>
struct funcionario{
    char nome[80];
    char cargo[10];
    float salario;
};
int main(){
    struct funcionario func;
    strcpy(func.nome, "Luana Batista");
    strcpy(func.cargo, "professora");
    func.salario = 2000.00;
    printf("Nome=%s, cargo=%s, salario=%f",
        func.nome, func.cargo, func.salario);
    return 0;
}
```

Registro de Registro

55

- Podemos ter também um registro dentro de outro registro. Isso é útil para modularizar melhor alguns campos dentro de um registro

```
struct endereco{
    char rua[30], cep [9], bairro[15], estado[2];
    int numero;
};

struct funcionario{
    char nome[80], cargo[10];
    float salario;
    struct endereco residencia;
};

...

struct funcionario func;
func.salario = 2000.00;
//acessar variáveis do endereço residência?

...
```

Registro de Registro

56

- Podemos ter também um registro dentro de outro registro. Isso é útil para modularizar melhor alguns campos dentro de um registro

```
struct endereco{
    char rua[30], cep [9], bairro[15], estado[2];
    int numero;
};

struct funcionario{
    char nome[80], cargo[10];
    float salario;
    struct endereco residencia;
};

...

struct funcionario func;
func.salario = 2000.00;
func.residencia.numero = 63;

...
```


Definição de “novos” tipos

57

- ❑ A linguagem C permite criar nomes de tipos:
 - Ex: typedef float Real
- ❑ Em geral, definimos nomes de tipos para as estruturas com as quais nosso programa trabalham. Por exemplo:

```
struct funcionario{  
    char nome[80];  
    char cargo[10];  
    float salario;  
};  
  
typedef struct funcionario Funcionario;
```

Definição de “novos” tipos

Exemplo

58

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

struct funcionario{
    char nome[80];
    char cargo[10];
    float salario;
};
typedef struct funcionario funcionario;

int main(){
    funcionario func;
    strcpy(func.nome, "Luana Batista");
    strcpy(func.cargo, "professora");
    func.salario = 2000.00;
    printf("Nome=%s, cargo=%s, salario=%f",
           func.nome, func.cargo, func.salario);
    return 0;
}
```

Definição de “novos” tipos

Exemplo


59

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

struct funcionario{
    char nome[80];
    char cargo[10];
    float salario;
};

typedef struct funcionario funcionario;

int main(){
    funcionario func;
    strcpy(func.nome, "Luana Batista");
    strcpy(func.cargo, "professora");
    func.salario = 2000.00;
    printf("Nome=%s, cargo=%s, salario=%f",
           func.nome, func.cargo, func.salario);
    return 0;
}
```



```
typedef struct{
    char nome[80];
    char cargo[10];
    float salario;
} funcionario;
```

Funções

Conceitos

Passagem de parâmetros

Função

61

- ❑ Função é um trecho de um algoritmo independente com um objetivo determinado, simplificando o entendimento do algoritmo e proporcionando ao algoritmo menos chances de erro e de complexidade
- ❑ Através da passagem de argumentos aos seus **parâmetros** e através de seu nome, uma função permite que seja retornado um valor a rotina chamadora

Parâmetros da função.

```
int maior(int a, int b){  
    int m;  
    if(a > b)  
        m = a;  
    else  
        m = b;  
    return m;  
}
```

Valor retornado pela função.

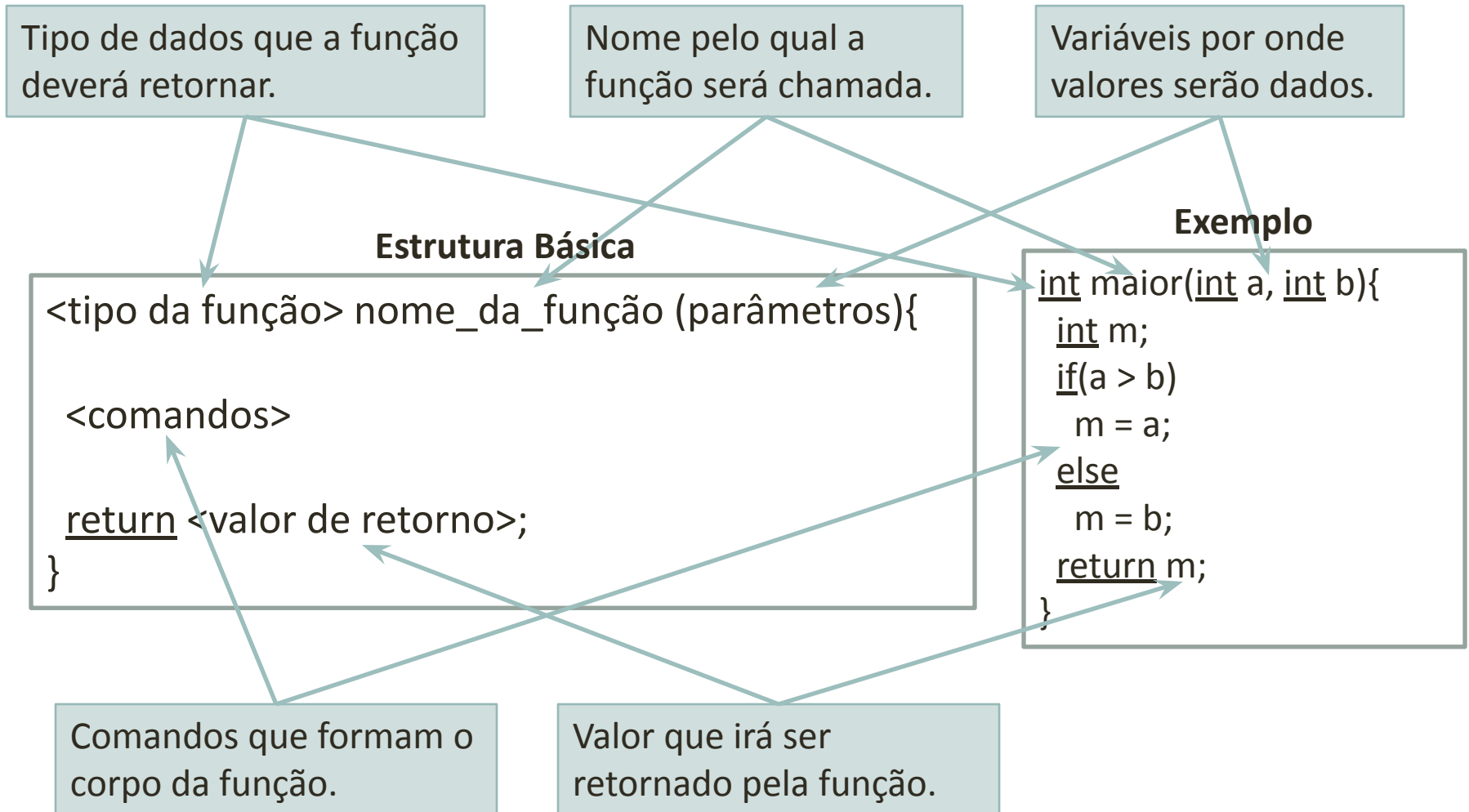
maior(8, 4) = 8

Valor retornado pela função.

Argumentos passados aos parâmetros da função.

Estrutura de uma Função

62



Chamada de Função

63

- Quando uma função é chamada, o fluxo de controle é desviado para a função. Ao terminar a execução dos comandos da função, o fluxo de controle retorna ao comando seguinte àquele onde a função foi ativada

```
int main() {  
    int c, d, e, f, g, h, i;  
    c = 2; d = 3;  
    e = maior(c, d);  
  
    f = 10; g = 7;  
    h = maior(f, g);  
  
    i = maior(6, 0);  
    return 0;  
}
```

```
int maior(int a, int b) {  
    int m;  
    if(a > b)  
        m = a;  
    else  
        m = b;  
    return m;  
}
```

Escopo de Variáveis e Funções

64

- ❑ O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa
- ❑ Tipos de escopos:
 - ❑ **Variáveis locais**
 - São declaradas em um bloco (função) e só tem validade dentro desse bloco
 - ❑ **Variáveis globais**
 - São declaradas fora de todas as funções do programa e tem validade em todas as funções
 - ❑ **Parâmetros formais**
 - São declarados como sendo as entradas (parâmetros) de uma função e só tem validade dentro dessa função

Função

Exemplo: maior

65

Variáveis locais

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

```
int c, d, e, f, g, h, i;
```

Variáveis globais

```
int maior(int a, int b){
```

Variáveis de parâmetros

```
    int m;
```

```
    if(a > b)
```

```
        m = a;
```

```
    else
```

```
        m = b;
```

```
    return m;
```

```
}
```

```
int main() {
```

```
    c = 2; d = 3;
```

```
    e = maior(c, d);
```

```
    f = 10; g = 7;
```

```
    h = maior(f, g);
```

```
    i = maior(6, 0);
```

```
    return 0;
```

```
}
```

Passagem de Parâmetros

66

- ❑ **Passagem de parâmetros por valor:**
 - A função recebe uma cópia do valor da variável que é fornecida quando é invocada. Todas as alterações feitas dentro da função não vão afetar os valores originais
- ❑ **Passagem de parâmetros por referência:**
 - Neste caso o que é enviado para a função é uma referência às variáveis utilizadas, e não uma simples cópia. As alterações realizadas dentro da função irão alterar os valores contidos nessas variáveis

Passagem de Parâmetros por Valor

67

```
#include <stdio.h>
int c, d;
void troca(int a, int b){
    int aux;
    aux = a;
    a = b;
    b = aux;
}
int main(){
    c = 2; d = 3;
    printf ("Antes: C=%d, D=%d \n", c, d);
    troca(c, d);
    printf ("Depois: C=%d, D=%d \n", c, d);
    return 0;
}
```



Passagem de Parâmetros por Valor

68

```
#include <stdio.h>
int c, d;
void troca(int a, int b){
    int aux;
    aux = a;
    a = b;
    b = aux;
}
int main(){
    c = 2; d = 3;
    printf ("Antes: C=%d, D=%d \n", c, d);
    troca(c, d);
    printf ("Depois: C=%d, D=%d \n", c, d);
    return 0;
}
```



Passagem de Parâmetros por Referência

69

```
#include <stdio.h>
int c, d;
void troca(int *a, int *b){
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
int main(){
    c = 2; d = 3;
    printf ("Antes: C=%d, D=%d \n", c, d);
    troca(&c, &d);
    printf ("Depois: C=%d, D=%d \n", c, d);
    return 0;
}
```

O * nos parâmetros indica que elas se tratam de **ponteiros**.

O & nas variáveis indica que será passado o **endereço de memória** da variável.



Passagem de Parâmetros por Referência

70

```
#include <stdio.h>
int c, d;
void troca(int *a, int *b){
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
int main(){
    c = 2; d = 3;
    printf ("Antes: C=%d, D=%d \n", c, d);
    troca(&c, &d);
    printf ("Depois: C=%d, D=%d \n", c, d);
    return 0;
}
```

O * nos parâmetros indica que elas se tratam de **ponteiros**.

O & nas variáveis indica que será passado o **endereço de memória** da variável.



Passagem de Parâmetros

Exemplo: operações de um quadrado

71

```
#include <stdio.h>
#include <math.h>
float area(float lado){
    return lado*lado; }
float perimetro(float lado){
    return 4*lado; }
float semiPerimetro(float lado){
    return perimetro(lado)/2; }
float diagonal(float lado){
    float d = 2*lado*lado;
    return sqrt(d); }
int main() {
    float lado = 4;
    printf("Area=f%", area(lado));
    printf("Perimetro=f%", perimetro(lado));
    printf("Semi perimetro=f%", semiPerimetro(lado));
    printf("Diagonal=f%", diagonal(lado));
    return 0;
}
```

Resumindo....

72



- Introdução a Linguagem C
 - Tipos básicos de variáveis
 - Operadores aritméticos, relacionais e lógicos
 - Comando de saída e entrada
 - Comandos de repetição
- Matrizes e vetores
 - Estruturas homogêneas
- Registros
 - Estrutura heterogênea
- Função
 - Passagem de parâmetros

Sugestão de leitura/estudo

73

- **Capítulo 1,2,3 e 4 do livro**
 - SCHILDT, Herbert. C completo e total. Makron, 3a edição revista e atualizada, 1997.
- <http://mtm.ufsc.br/~azeredo/cursoC/c.html>
- <http://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/c.pdf>

Atividade de fixação

74

- Faça uma função chamada "media" que recebe um vetor double, um inteiro "n" que indica o tamanho do vetor e um inteiro "i" passado por referência. A função deve retornar a média dos n elementos do vetor e o inteiro i, passado por referência, deve retornar à posição (índice) do elemento que possui o valor mais próximo da média.

Assinatura: media(double vet[], int n, int* i)

ex: vetor = 10, 20, 30, 40, 50, 2 => $152/6 = 25,333..$ | índice = 2

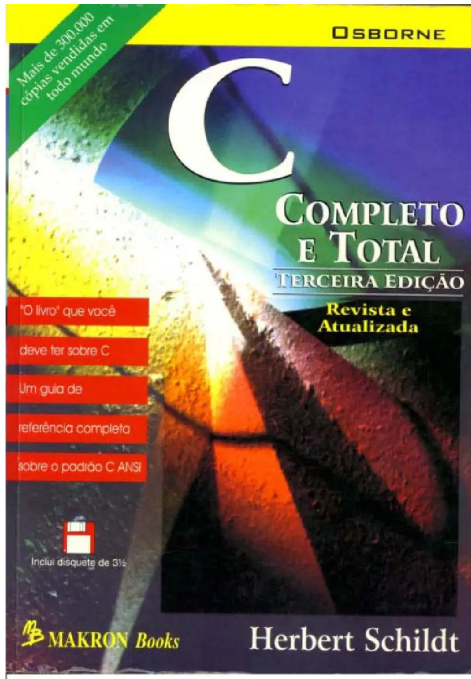
- Escreva uma função chamada "fatoraPotencia" que recebe um valor "numero" passado por valor e dois inteiros "base" e "expoente" passados por referência. Sua função deve descobrir quais valores "base" e "expoente" devem assumir tal que a condição $base^{\text{expoente}} = \text{numero}$ e a base tenha o menor valor possível.

Assinatura: void fatoraPotencia(int numero, int *base, int *expoente)

ex: numero = 16 | base = 2 e expoente = 4 => 16

Referências

75



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.