Lógica de programação

Variáveis e entrada de dados

Walisson Pereira

walisson_pereira@uvanet.br
Universidade Estadual Vale do Acaraú

Roteiro

Disclaimer

Disclaimer

Todo o código em linguagem C precisa seguir a seguinte estrutura básica:

```c
#include <stdio.h>
int main() {
  //TODO: algoritmo codificado em linguagem C
}
```

Devido as limitações de espaço no slide, eventualmente, será apresentado apenas o trecho de código principal. Para realizar o teste do código em um computador, é necessário incluir o trecho de código onde está o comentário (trecho em verde) no exemplo acima.

Disclaimer

O código-fonte deve ser gerado em um editor de texto puro (gedit, bloco de notas, etc.)

No terminal linux, o comando gcc é usado para compilar um código-fonte em linguagem C.

```
gcc codigo.c
./a.out
```

O arquivo executável padrão criado é o **a.out** e para executá-lo, deve-se incluir o **./** antes.

Usando os comandos abaixo:

```
gcc codigo.c -o executavel
./executavel
```

A opção -**o** é usada para nomear o arquivo executável.

Variáveis

Variáveis

Nome de variáveis devem iniciar obrigatoriamente com uma letra, mas podem conter números e o símbolo sublinha (_).

Variáveis

Exemplos de nomes de variáveis

- ✓ a1
- ✓ velocidade
- ✓ velocidade90
- X salário_médio
- X salario medio
- ✓ salario_medio
- ✓ _b
- X 1a

Variáveis

As linguagens de programação geralmente classificam os dados da seguinte forma:

- **Inteiro:** são os números pertencentes ao conjunto de números inteiros ($\in \mathbb{Z}$).
  Exemplo: 15, 0, 9, -3
- **Ponto flutuante:** são os números pertencentes ao conjunto de números reais ($\in \mathbb{R}$).
  Exemplo: 1.67, 0.00, -2.45

Variáveis

As linguagens de programação geralmente classificam os dados da seguinte forma:

- **Caractere:** é formado por uma letra ou um símbolo, geralmente, escrito entre aspa simples.
  Exemplo: 'a', '2', '@'.
- **String:** é formados por um conjunto de caracteres alfanuméricos, geralmente, escrito entre aspas dupla.
  Exemplo: "A menina é bonita".
- **Booleano (lógico):** é formado por um conjunto com duas possibilidades.
  Exemplo: verdadeiro ou falso.

Tipos numéricos

- Todos os números são representados internamente utilizando o sistema binário.
- Usamos a palavra **int** para declarar que uma variável guarda um valor inteiro.
- Usamos a palavra **float** para declarar que uma variável guarda um valor em ponto flutuante.

Tipos numéricos

Quando usado apenas variáveis do tipo `int`, o computador fará os cálculos usando apenas este tipo de representação.

```
1 #include <stdio.h>
2 int main() {
3   int n1, n2, resultado;
4   n1 = 10;
5   n2 = 4;
6   resultado = n1 / n2;
7   printf("%d\n", resultado);
8 }
```

Observe que o resultado do código acima é um inteiro.

Tipos numéricos

Trecho de código com exemplo de operações matemática com inteiros:

```
1  int x, y, a, b, c, d, e, f;
2  x = 10;
3  y = 3;
4  a = x + y;  // a recebe 13
5  b = x - y;  // b recebe 7
6  c = x * y;  // c recebe 30
7  d = x / y;  // d recebe 3
8  e = x % y;  // e recebe 1
9  f = -x;     // f recebe -10
```

Variáveis

Quando desejamos trabalhar com números reais, usamos as variáveis do tipo `float`.

```
1  #include <stdio.h>
2  int main() {
3    float n1, n2, resultado;
4    n1 = 10;
5    n2 = 4;
6    resultado = n1 / n2;
7    printf("%f\n", resultado);
8  }
```

Observe que o resultado do código acima é um número fracionário

Tipos numéricos

Trecho de código com exemplo de operações matemática com ponto flutuante:

```
1  float p, q, x, z;
2  int y;
3
4  x = 10;
5  y = 4;
6  p = 50 / 30;    // p recebe 1.0
7  q = 10.0 / 4;   // q recebe 2.5
8  z = x / y;      // z recebe 2.5
```

Variáveis

Algumas curiosidades:

- A linguagem C é baseada na língua inglesa, por isso, um número fracionário não é representado como 2,5 e sim 2.5.
- Um número do tipo float é guardado na memória em formato de notação científica, porém é baseado na base 2.

Tipo booleano (lógico)

- Apenas dois valores são permitidos:
  - True (Verdadeiro)
  - False (Falso)

  Entretanto, não existe este tipo booleano nativamente em C. Para efeito de operações lógicas, a variável é um inteiro onde:
  - True é qualquer valor diferente de zero
  - False é o valor zero

Tipo booleano (lógico)

- Operadores relacionais

Operador	Operação	Símbolo matemático
==	igualdade	$=$
>	maior que	$>$
<	menor que	$<$
!=	diferente	$\neq$
>=	maior ou igual	$\geq$
<=	menor ou igual	$\leq$

Tipo booleano (lógico)

Experimente:

```c
#include <stdio.h>
int main() {
  int a, b, c, d, resultado;
  a = 1;
  b = 5;
  c = 2;
  d = 1;
  resultado = a == b;
  printf("%d\n", resultado);
}
```

Tipo booleano (lógico)

Experimente substituir a linha 8 do código anterior:

```
8 resultado = a < b;
```

```
8 resultado = a != b;
```

```
8 resultado = a < b;
```

```
8 resultado = a == d;
```

```
8 resultado = b >= a;
```

```
8 resultado = c <= b;
```

```
8 resultado = d != a;
```

```
8 resultado = d != b;
```

Tipo booleano (lógico)

Exercício:

Antes de testarem no computador, responda o exercício considerando as variáveis:

```
1 int a = 4, b = 10, d = 1, f = 5;
2 float c = 5.0;
```

Qual o resultado das seguintes expressões lógicas?

- $a == c$
- $a < b$
- $d > b$
- $c != f$
- $a == b$
- $c < d$
- $b > a$
- $c >= f$
- $f >= c$
- $c <= c$
- $c <= f$

Tipo booleano (lógico)

- Operadores lógicos

Operador lógico	Operador em C	Operação
$\neg$	!	não (negação)
$\wedge$	&&	e (conjunção)
$\vee$	||	ou (disjunção)

Tipo booleano (lógico)

- Operador lógico de **negação** (!)

V_1	**not** V_1
V	F
F	V

Tipo booleano (lógico)

- Operador lógico **e** (&&)

V_1	V_2	V_1 **and** V_2
F	F	F
F	V	F
V	F	F
V	V	V

Memorize: Só é verdade se todas as entradas forem verdadeiras.

Tipo booleano (lógico)

- Operador lógico **ou** (||)

V_1	V_2	V_1 **or** V_2
F	F	F
F	V	V
V	F	V
V	V	V

Memorize: Só é falso se todas as entradas forem falsas.

Tipo booleano (lógico)

Exercício:

Antes de testar no computador, responda o exercício considerando que:

```
1 int a = 1, b = 0, c = 1;
```

Qual o resultado das seguintes expressões lógicas?

- a && c
- b && b
- !c
- !b
- !a
- a || c
- b || c
- c || a
- c || b
- c || c
- a && b
- b || b
- b && c

Tipo booleano (lógico)

- Os operadores lógicos podem ser combinados em expressões lógicas mais complexas.
- A resolução da expressão lógica é feita seguindo a seguinte prioridade:
  1. !
  2. &&
  3. ||

  Qual seria o resultado?

```
1 || 0 && !1
```

Tipo booleano (lógico)

Exemplos:

```c
#include <stdio.h>
int main() {
  float salario = 1000.00;
  int idade = 20;
  int pode_contratar = salario > 1000 && idade > 18;
  printf("%d\n", pode_contratar);
}
```

```c
#include <stdio.h>
int main() {
  float salario = 2000.00;
  int idade = 30;
  int pode_contratar = salario > 1000 && idade > 18;
  printf("%d\n", pode_contratar);
}
```

Tipo booleano (lógico)

Exercício:

Antes de testar no computador, responda qual é o resultado das seguinte expressão lógica?

```
1 a > b && c || d
```

Considere os seguintes casos:

	a	**b**	**c**	**d**
a)	1	2	1	0
b)	10	3	0	0
c)	5	1	1	1

Memorize:

As operações matemática e lógicas são realizadas de acordo com a seguinte ordem de precedência:

1. Parênteses
2. Multiplicação (*), divisão (/) e módulo (%)
3. Adição (+) e subtração (-)
4. Expressões lógicas:
   5.1 Relacionais (==, ! =, <, >, <= e >=)
   5.2 !
   5.3 &&
   5.4 ||

As operações de mesma prioridade são realizadas da esquerda para a direita.

Curiosidade sobre a linguagem C

Nos exercícios de matemática, para verificar se uma variável `x` está entre o intervalo de 0 a 10 (exclusivo), as operações relacionais podem ser feito neste formato:

```
1 0 < x < 10
```

Entretanto, na linguagem C, é necessário fazer as operações neste formato:

```
1 0 < x && x < 10
```

Porém, se você usar o primeiro formato em C, o compilador não acusará erro de sintaxe, mas o resultado não será o esperado. Tente executar com `x` sendo um valor negativo.

Tipo char

Variáveis do tipo **char** armazenam a informação de um único caractere.

Exemplo de uso:

```
1 char letraA, letraFMaiuscula, simboloSoma;
2
3 letraA = 'a';
4 letraFMaiuscula = 'F';
5 simboloSoma = '+';
6
7 printf("%c %c %c\n", letraA, letraFMaiuscula, simboloSoma);
```

Observe que um caractere sempre é representado entre aspas simples.

Tipo string

Variáveis do tipo **string** armazenam cadeias de caracteres como nomes e textos em geral.

Na linguagem C, não há um tipo específico para string. Por isso, representamos uma string como um vetor de caracteres.

Exemplo de uso:

```
1 char nome[20] = "Joaozinho joga bola";
2
3 printf("%s\n", nome);
```

Observe que um caractere sempre é representado entre aspas duplas.

Tipo string

- **Cadeia de caracteres** é uma sequência de símbolos como letras, números, sinais de pontuação, etc.

  Exemplo: ”Joaozinho joga bola”.

J	o	a	z	i	n	h	o		j	o	g	a		b	o	l	a

Tipo string

É possível acessar o conteúdo de uma string através de seu índice

0	1	2	3	4	5	6	7	8	← Índice
J	o	a	o	z	i	n	h	o	← Conteúdo

Faça o teste:

```
1  char nome[10] = "Joaozinho";
2
3  printf("%s\n", nome);
4
5  printf("%c\n", nome[0]);
6  printf("%c\n", nome[1]);
7  printf("%c\n", nome[2]);
8  printf("%c\n", nome[3]);
```

Saída da dados

Tipo string

O comando de saída de dados serve para escrever mensagens e exibir valores de expressões, proporcionando mais informação, tanto para o usuário quanto para o próprio programador.

A sintaxe para o uso do comando de saída de dados é a seguinte:

```
1 printf("<frmt1><frmt2> ... <frmtN>", exp1, exp2, ..., expN);
```

Tipo string

A tabela apresenta alguns formatos comuns aceito pelo comando **printf**

Código	**Significado**
%c	Caractere
%d	Inteiros decimais com sinal
%e	Notação científica
%f	Ponto flutuante decimal
%s	String
%%	Escreve o símbolo "%"

Tipo string

Trecho de código com exemplo de uso do comando **printf**

```
1 printf("Alo mundo!");
2 printf("Tudo bem com voces");
```

```
1 float saldo1 = 100.00;
2 float saldo2 = 200.00;
3 float soma = saldo1 + saldo2;
4 printf("%f\n", soma);
```

Tipo string

Os marcadores também servem para formatar como queremos que a informação seja escrita.

Faça o teste:

```
1 int idade = 22;
2 char nome[20] = "Maria";
3 printf("%s tem %d anos\n", nome, idade);
```

Tipo string

Os marcadores também são úteis para formatar números decimais.

Faça o teste:

```
1 int idade = 22;
2 char nome[20] = "Maria";
3 float grana = 3.50;
4 printf("%s tem %d anos e apenas R$ %.2f no bolso\n", nome, idade,
      grana);
```

Entrada de dados

Entrada de dados

É comum que programas recebam dados do meio externo.

Exemplo:

- do usuário através do teclado ou mouse;
- do disco através da leitura de arquivos;
- da rede através de uma conexão.

Tipo string

O comando de entrada de dados serve para captar do usuário do programa um ou mais valores necessários para a execução das tarefas.

O comando **scanf** é usado para solicitar dados do usuário pelo teclado.

A sintaxe para o uso do comando **scanf** é a seguinte:

```
1 scanf("<frmt1><frmt2> ... <frmtN>", &var1, &var2, ..., &varN);
```

Atenção: observe que nos parâmetros do **scanf**, o símbolo **&** é usado como prefixo do nome das variáveis.

Tipo string

A tabela apresenta alguns formatos comuns aceito pelo comando **scanf**

Código	**Significado**
%c	Lê um único caractere
%d	Lê um inteiro
%f	Lê um número em ponto flutuante
%s	Lê uma string até a digitação do espaço em branco

Tipo string

Trecho de código com exemplo de uso do comando **scanf**:

```
1 int numero;
2 scanf("%d", &numero);
3 printf("%d\n", numero)
```

Tipo string

Trecho de código com exemplo de uso do comando **scanf**:

```
1 int numero1, numero2, diferenca;
2 scanf("%d %d", &numero1, &numero2);
3 diferenca = numero1 - numero2;
4 printf("%d\n", diferenca)
```

Tipo string

Trecho de código com exemplo de uso do comando **scanf**:

```
1 float nota1, nota2, media;
2 printf("Digite a primeira nota: ");
3 scanf("%f", &nota1);
4 printf("\nDigite a segunda nota: ");
5 scanf("%f", &nota2);
6 media = (nota1 + nota2) / 2;
7 printf("\nSua media e %.1f\n", media)
```

Entrada de dados

Trecho de código com exemplo de uso do comando **scanf**:

```
1 char nome[30];
2 scanf("%s", nome);
3 printf("Seu nome: %s\n", nome);
```

Perceberam alguma coisa diferente neste código?

Erros comuns

- A entrada de dados é um ponto frágil em nosso programas.
- Não temos como prever o que o usuário vai digitar, então, temos que nos preparar para reconhecer os erros mais comuns.

Erros comuns

Faça o teste: escreva um programa com o seguinte trecho de código.

```
1  char nome[30];
2  int idade;
3  float saldo;
4  printf("Digite o seu nome: ");
5  scanf("%s", nome);
6  printf("\nDigite a sua idade: ");
7  scanf("%d", &idade);
8  printf("\nDigite o saldo da sua conta bancria: ");
9  scanf("%f", &saldo);
10 printf("Seu nome e %s e tem %d anos.\nSua conta tem R$ %.2f",
       nome, idade, saldo);
```

Erros comuns

Experimente digitar os valores seguintes a cada execução:

- João · 42 · 15756.34
- Maria · 28 · 34
- Minduim · abc · 34
- Juanito · 31 · abc
- Mary · 25 · 17,4

Leia atentamente e procure entender qual o erro foi identificado pelo compilador.

Sequência e tempo

Sequência e tempo

A sequência dos comandos importa.

Exemplo: a variável **pedido** indica o valor de um pedido em um restaurante e a variável **total** indica o valor a ser pago no final da refeição.

```
1  #include <stdio.h>
2  int main() {
3    int total = 0;
4    int pedido = 100;
5    total = total + pedido;
6    pedido = 200;
7    total = total + pedido;
8    total = total + pedido;
9    pedido = 0;
10   printf("%d\n", total);
11 }
```

Sequência e tempo

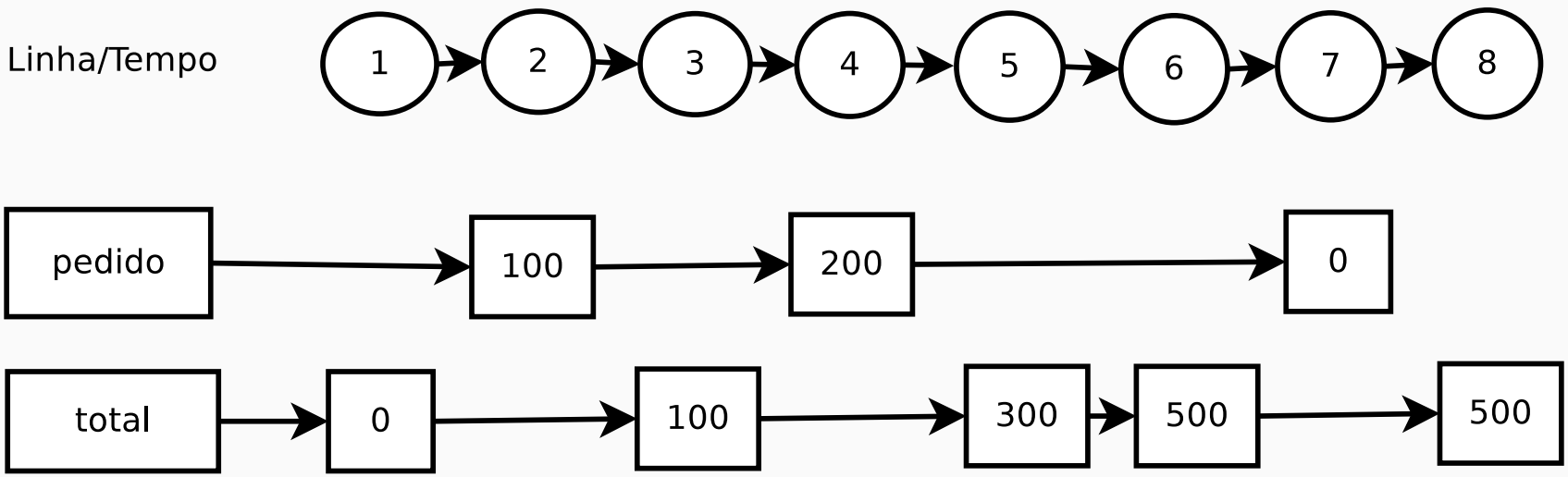

Figura 1: Mudança no valor das variáveis no tempo [1]

Rastreamento

A diferença entre ler um texto e um programa é justamente seguir a mudança de valores de cada variável conforme o programa é executado.

Rastreamento

- Entender que o valor das variáveis pode mudar durante a execução do programa não é tão natural, mas **é fundamental** para a programação de computadores.
- **É fundamental** verificar linha a linha os efeitos e mudanças causados no valor de cada variável.

Rastreamento

- Não esqueça que para programar corretamente você precisa ser capaz de **entender** o que cada linha do programa significa e os **efeitos** que ela produz.

Rastreamento

- O **rastreamento** é importante para entender o código e encontrar erros.
- É um processo detalhado que precisa de **atenção**.
- **Não tente** simplifica-lo ou começar a rastrear no meio de um programa.
- Você deve rastrear **linha a linha**, do início ao fim do programa.
- Se encontrar um **erro**, **pare** e o **corrija**, então **recomece** o rastreamento do início.

Rastreamento

- **É essencial** que você domine a atividade do rastreamento.
- O rastreamento que vai lhe responder o que o programa **faz de fato** e ele é essencial para **entender seu funcionamento**.
- **Programar é detalhar**, e que simplesmente ler o texto de um programa **não é suficiente**.

Rastreamento

- O rastreamento é a **melhor ferramenta** para descobrir o que está acontecendo no seu programa.
- Embora pareça óbvio, esse é **um dos erros mais comuns** quando se começa a programar.

Rastreamento

Exercício:

Faça o rastreamento do código a seguir.

```c
#include <stdio.h>
int main() {
  int total = 0;
  int pedido = 100;
  total = total + pedido;
  total = 200;
  total = total + pedido;
  pedido = 500;
  total = total + pedido;
  printf("%d\n", total);
}
```

Referências

Referências

1 ASCENCIO, A.F.G.; CAMPOS, E.A.V. de. Fundamentos da Programção de Computadores. Algoritmos, Pascal e C/C++. São Paulo: Pearson Prentice Hall, 2012.

2 VAREJÃO, F. M. V. Introdução à programação: uma nova abordagem usando C. Rio de Janeiro: Elsevier, 2015.

3 BACKES, A. Linguagem C: completa e descomplicada. Rio de Janeiro: Elsevier, 2013.