Programação em C, variáveis, tipos, operadores, E/S

Rafael Beserra Gomes

UFRN

Material compilado em 13 de dezembro de 2017.

Programando em C

- Desenvolvimento inicial por Dennis Ritchie por volta de 1970
- Em 1983, um comitê foi criado pela ANSI (American National Standards Institute) para padronizar a linguagem C (**ANSI C**)
- Alguns dos padrões:
  - **C89** (retificações em **C90**) e **ISO C** (ISO9899:1990)
  - ISO C atualizado em 1999: ISO/IEC 9899:1999 (ou **C99**)
  - ISO C atualizado em 2011: ISO/IEC 9899:2011 (ou **C11**)
- O compilador **gnu/gcc** pode incluir extensões da linguagem

- Desenvolvimento inicial por Dennis Ritchie por volta de 1970
- Em 1983, um comitê foi criado pela ANSI (American National Standards Institute) para padronizar a linguagem C (**ANSI C**)
- Alguns dos padrões:
  - **C89** (retificações em **C90**) e **ISO C** (ISO9899:1990)
  - ISO C atualizado em 1999: ISO/IEC 9899:1999 (ou **C99**)
  - ISO C atualizado em 2011: ISO/IEC 9899:2011 (ou **C11**)
- O compilador **gnu/gcc** pode incluir extensões da linguagem

1. As extensões adicionam mais recursos à linguagem, mas não é padrão! Isso significa que o seu código poderá não ser compilado em todos os sistemas.

Programando em C

Estrutura **inicial** de um programa escrito em C:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5
6      printf("Mensagem escrita na tela\n");
7      printf("Uma outra mensagem\n");
8
9      return 0;
10 }
```

- **Cabeçalho**
- Definição de uma função principal chamada **main**: instruções e retorno

Programando em C

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

int main() {

    printf("Mensagem escrita na tela\n");
    printf("Uma outra mensagem\n");

    return 0;
}
```

- Instruções são encerradas com **;**
- Quebras de linha e **indentação** facilitam a legibilidade do código-fonte
- Comentários podem ser feitos com:
  - // para uma única linha (a partir de C99)
  - /* ... */ para múltiplas linhas

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

int main() {

    printf("Mensagem escrita na tela\n");
    printf("Uma outra mensagem\n");

    return 0;
}
```

- Instruções são encerradas com **;**
- Quebras de linha e **indentação** facilitam a legibilidade do código-fonte
- Comentários podem ser feitos com:
  - // para uma única linha (a partir de C99)
  - /* ... */ para múltiplas linhas

1. A indentação refere-se ao espaçamento em relação à margem. Acostume-se a manter uma boa indentação desde o começo.
2. Esquecer ; é um erro bastante comum

Programando em C

Compilando

```
1 gcc codigo.c -o nomeArquivoExecutavel
```

Há várias opções para o compilador gcc, algumas:

- -std: especifica o padrão da linguagem C a ser utilizada na compilação. Por exemplo: -std=c99
- -pedantic: alerta se há algo fora do padrão ISO C ou do padrão especificado por -std
- -Wall: habilita todos os alertas sobre possíveis erros de programação
- -O3: habilita algumas otimizações de código

Compilando

```
1 gcc codigo.c -o nomeArquivoExecutavel
```

Há várias opções para o compilador gcc, algumas:

- -std: especifica o padrão da linguagem C a ser utilizada na compilação. Por exemplo: -std=c99
- -pedantic: alerta se há algo fora do padrão ISO C ou do padrão especificado por -std
- -Wall: habilita todos os alertas sobre possíveis erros de programação
- -O3: habilita algumas otimizações de código

1. Se você omitir o -o, o executável terá o nome a.out por padrão
2. Incluir -Wall é útil para evitar certos erros de programação
3. Incluir -pedantic pode alertá-lo se seu código estiver fora do padrão

IDE

IDE: *integrated development environment* auxiliam a programação de computadores

- Texto colorido para melhorar a legibilidade do código
- Atalhos para compilação, acesso fácil para configurações
- Recursos para refatoração, debug, etc

O que posso usar?

- Para C: Dev-cpp, Code::Blocks, Sun Studio, Eclipse, Sublime, MS Visual Studio, etc

Cuidado: as facilidades podem ocultar detalhes importantes do processo de programação

Variáveis e tipos

Revendo a representação dos dados na memória

Exemplo de dados na memória:

0xbffff22c	0	0	0	0	0	1	0	1	5 **inteiro curto**
0xbffff22d	0	1	0	0	0	0	1	0	B **caractere**
0xbffff22e	0	1	0	0	0	0	1	1	C **caractere**
0xbffff22f	1	1	0	1	1	1	0	1	3.2 **real**
0xbffff230	1	1	0	0	1	1	0	0	
0xbffff231	0	1	0	0	1	1	0	0	
0xbffff232	0	1	0	0	0	0	0	0	
0xbffff233	0	1	0	0	0	0	0	1	A **caractere**
0xbffff234	0	1	0	0	0	0	1	0	B **caractere**
0xbffff235	0	1	0	0	0	0	1	1	C **caractere**
0xbffff236	0	1	0	0	0	0	1	1	A **caractere**
0xbffff237	1	1	1	1	1	0	1	1	-5 **inteiro curto**
0xbffff238	0	0	0	0	0	0	0	0	1 **inteiro**
0xbffff239	0	0	0	0	0	0	0	0	
0xbffff23a	0	0	0	0	0	0	0	0	
0xbffff23b	0	0	0	0	0	0	0	1	

Endereço na memória (em hexadecimal)

Variáveis

- A variável representa um dado variável (pode ser alterado) na memória
- Cada variável possui um **tipo** (ex.: int, float, string, boolean)
- Cada variável possui um **endereço**, que representa a posição do dado na memória, na qual há uma sequência de *bits*[1] representando um **valor**
- Cada variável possui um **identificador**, evitando que tenhamos que saber o endereço dos dados

[1] a quantidade de bits depende do tipo da variável

Variáveis

Em relação a C:

- é necessário **declarar** variáveis

```
1  #include <stdio.h>
2
3  int main() {
4
5      int x;
6      float y;
7      int z = 323;
8      int a, b = 2, c;
9
10     return 0;
11 }
```

- até C90 todas as declarações de variáveis devem estar no início da função main
- pode atribuir um valor inicial para cada variável (**inicialização**)
- pode declarar mais de uma variável na mesma linha, desde que sejam do mesmo tipo

Em relação a C:

- é necessário **declarar** variáveis

```c
#include <stdio.h>

int main() {

    int x;
    float y;
    int z = 323;
    int a, b = 2, c;

    return 0;
}
```

- até C90 todas as declarações de variáveis devem estar no início da função main
- pode atribuir um valor inicial para cada variável (**inicialização**)
- pode declarar mais de uma variável na mesma linha, desde que sejam do mesmo tipo

1. Uma boa regra é procurar sempre inicializar as variáveis. Apesar de que em testes você poderá constatar um valor zero inicial para as variáveis, nem sempre será o caso.

Variáveis

- Experimente o seguinte código:

```c
#include <stdio.h>

int main() {

    int a = 3;
    printf("Valor da variavel a: %d\n", a);
    printf("Endereco da variavel a: %p\n", &a);
    a = a + 2;
    printf("Valor da variavel a: %d\n", a);

    return 0;
}
```

Tipos

Em relação a C:

- tipos primitivos principais: char, int, long long, float, double

Tipo	**Exemplo de valor**
char[2]	'a', 'z', 'A', 'T' (tabela ascii)
int	1232, 502, -39328
long long int	2395828482, -2392542832
float	3.242, 52002.1
double	3.242, 52002.1

- as versões unsigned utilizam a variável somente para valores não negativos
- execute o código limites.c disponível na página

[2] um caractere tem valor inteiro de acordo com a tabela ascii

Tipos

Strings são sequências de caracteres

- uma string é representa entre aspas duplas

```
1 "eis uma string"
```

- é possível armazenar uma string em C, mas veremos mais adiante na disciplina

Tipos

Um **overflow** ocorre quando um valor ultrapassa os limites máximos para aquele tipo

- Tente causar um overflow no programa somaDoisNumeros.c disponível na página

```c
#include <stdio.h>

int main() {

    int a, b;

    scanf("%d %d", &a, &b);
    printf("%d\n", a+b);

    return 0;
}
```

Um **overflow** ocorre quando um valor ultrapassa os limites máximos para aquele tipo

- Tente causar um overflow no programa somaDoisNumeros.c disponível na página

```c
#include <stdio.h>

int main() {

    int a, b;

    scanf("%d %d", &a, &b);
    printf("%d\n", a+b);

    return 0;
}
```

1. Para trabalhar com números inteiros maiores o uso de bibliotecas é necessário. Infelizmente não há uma solução na biblioteca padrão. Em algumas linguagens de programação, como Python, os limites são adaptáveis. Entretanto, deve-se ter em mente que para trabalhar com números com representação maior que 64 bits, são necessários vários ciclos de clock para a realização das operações aritméticas simples como soma e multiplicação (e esse custo computacional fica oculto para o programador).

Identificadores

Identificadores em C:

- começam com não-dígito seguido por uma combinação (nula ou não) de não-dígitos ou dígitos
- não-dígitos: "a"..."z", "A"..."Z", _ ou *universal character name*
- dígitos: "0"..."9"
- não pode ser uma palavra-chave (ex.: int, return, float, if, for)
- são *case-sensitive*[3]

Recomendações:

- iniciar com letra minúscula para variáveis
- utilize um identificador que represente bem o significado da variável
- se for composto por vários nomes, comece cada nome (exceto o primeiro) com letra maiúscula (ex.: anguloTeste)
- identificadores longos demais são difíceis de ler (ex.: valorInteiroASerDigitadoPeloUsuario)

[3] o compilador diferenciará letras maiúsculas de letras minúsculas

Entrada e saída de dados

Saída de dados

- A saída de dados é realizada através da função **printf** (stdio.h):

```
1 printf("Uma mensagem\n");
2 printf("Estou programando");
3 printf("em C\n");
```

- É comum dizer que o programa **escreve na tela**
- Você pode inserir valores na string a ser escrita:

```
1 printf("Mes = %d e Ano = %d\n", 2, 2017);
```

Saída de dados

- haverá tantos **argumentos** quantos **especificadores de formato** na string (%), atribuídos na mesma ordem

```
1 printf("Dia = %d, Mes = %d e Ano = %d\n", 21, 2, 2017);
```

- o tipo do dado determina que **especificador de formato** utilizar

```
1 printf("%d de %s, juros = %f\n", 21, "Fevereiro", 0.13);
```

Acesse essa página para conhecer os especificadores de formato da linguagem C.

- para escrever na tela números com tantas casas de precisão:

```
1 printf("%.02f", 3.14159265359);
```

Saída de dados

Exercício em sala

Escreva um programa em C que declara as seguintes variáveis:

- um **caractere** com valor 'g' (identificador simboloGravidade)
- um número **real** com valor 9.8196 (identificador gravidade)

Em seguida, o programa deve, utilizando o caractere e o número real declarados, escrever na tela a seguinte mensagem:
O valor da gravidade g é: 9.8196 m/s2

Entrada de dados

- A entrada é realizada através da função **scanf** (stdio.h):

```
1 int idade;
2 printf("Digite a sua idade: ");
3 scanf("%d", &idade);
```

- É comum dizer que o programa **lê do usuário**

Entrada de dados

- haverá tantos **argumentos** quantos **especificadores de formato** na string (%), atribuídos na mesma ordem
- o **argumento** é um endereço de memória (usamos o operador & para obter o endereço de uma variável)

```
1 int idade;
2 printf("Digite a sua idade: ");
3 scanf("%d", &idade);
```

- o tipo do dado determina que **especificador de formato** utilizar

```
1 int idade;
2 float altura;
3 printf("Digite a sua idade: ");
4 scanf("%d", &idade);
5 printf("Digite a sua altura: ");
6 scanf("%f", &altura);
```

Entrada de dados

- você pode utilizar o mesmo scanf para ler mais de um dado

```
1 int idade;
2 float altura;
3 printf("Digite a sua idade e altura: ");
4 scanf("%d %f", &idade, &altura);
```

Entrada de dados

- a mensagem que precede o scanf **não é obrigatório**, serve apenas para orientar o usuário

```
1 int idade;
2 float altura;
3 scanf("%d %f", &idade, &altura);
```

- o scanf detecta o final de uma entrada para o começo da seguinte a partir de qualquer combinação de espaços, quebras de linha (’\n’) ou tabs (’\t’)

- a mensagem que precede o scanf **não é obrigatório**, serve apenas para orientar o usuário

```
1 int idade;
2 float altura;
3 scanf("%d %f", &idade, &altura);
```

- o scanf detecta o final de uma entrada para o começo da seguinte a partir de qualquer combinação de espaços, quebras de linha ('\n') ou tabs ('\t')

1. Nos exemplos das listas de exercícios e nos exemplos que devem constar nas provas, tudo o que o usuário digitar, ou seja, fizer parte da entrada, será destacado em **negrito**. Por exemplo:
   Quantos pontos possui o time: **50**
   Quantas partidas restam: **1**
   Quantos pontos são necessários: **54**
   Não é possível se classificar.
2. Nas listas de exercícios e nas provas, a não ser que expresso o contrário, você pode assumir que o usuário digita as informações conforme esperado. Por exemplo, para um programa que determina se um número inteiro é primo ou não, você pode assumir que ele digitará um número inteiro e maior que 0.

Entrada de dados

Exercício em sala

Escreva um programa em C que declara as seguintes variáveis:

- um **caractere** com valor 'g' (identificador simboloGravidade)
- um número **real** com valor 9.8196 (identificador gravidade)
- um número **inteiro** com o tempo (identificador tempo)

O programa deve escrever na tela a mensagem **"Digite o tempo de queda: "** e ler do usuário o tempo. Em seguida, deve escrever na tela a seguinte mensagem:
O valor da gravidade g é: 9.8196 m/s2
O espaço percorrido pelo objeto em queda livre foi: ... metros
Utilize o seguinte para calcular o espaço percorrido:

$$\frac{gt^2}{2}$$

Operadores

Tipos de operadores

Realizam alguma funcionalidade específica com os operandos.
Algumas categorias de operadores comuns em linguagens de programação:

- Operadores aritméticos
- Operadores de atribuição
- Operadores relacionais
- Operadores lógicos
- Operadores de deslocamento de bits (ainda não por enquanto :-)

Operadores aritméticos

Operadores
Operadores aritméticos

Operadores aritméticos

Operadores aritméticos

- +, -, *, /, %
- todos os operadores acima são binários (dois operandos numéricos)
- Exemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5      printf("%d\n", 2+3);
6      printf("%d\n", 5*3);
7      printf("%d\n", 5/2);
8      printf("%d\n", 17%3);
9
10     return 0;
11 }
```

Operadores de atribuição

Operadores
Operadores de atribuição

Operadores de atribuição

Operação para atribuir um valor a uma variável. Sintaxe:

- *<variável>* = *<expressão>*
- *<variável>* += *<expressão>*
- *<variável>* -= *<expressão>*
- *<variável>* *= *<expressão>*
- *<variável>* /= *<expressão>*
- *<variável>* %= *<expressão>*

Operadores de atribuição

Experimente:

```
1  #include <stdio.h>
2
3  int main() {
4
5      int x;
6      x = 3;
7      printf("Valor da variavel x: ", x);
8      x = 5;
9      printf("Valor da variavel x: ", x);
10     x -= 1;
11     printf("Valor da variavel x: ", x);
12
13     return 0;
14 }
```

Operadores lógicos

Operadores
Operadores lógicos

Lógica Proposicional

Lógica Proposicional

Sistema formal (modelo matemático que reflete um sistema abstrato de pensamentos) que possui como base **proposições** (expressas em sentenças) e **conectivos lógicos** (operadores) definidos por regras.

Exemplo de uma fbf (**fórmula bem formada**):

$$\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$$

onde p e q são proposições

Exemplos de proposição

- Uma proposição é avaliada como **verdadeira** ou **falsa**
- Exemplos de proposições (pela lógica aristotélica):
  - p : O céu é azul
  - q : Palmeiras é o melhor time da América do Sul
  - r : O Brasil é um país da América do Sul
  - s : Hoje choveu bastante

  Exemplos de sentenças que não são proposições:
  - t : Está chovendo lá fora?
  - u : Um evento fora da cidade

Lógica Proposicional

Conectivos lógicos

- negação: $\neg$ p
- conjunção (e): p $\wedge$ q
- disjunção (ou): p $\vee$ q
- implicação (se/então): p $\rightarrow$ q
- bi-implicação (se e somente se): p $\leftrightarrow$ q

Lógica Proposicional

Tabela verdade

Tabela verdade

Tabela que exibe todos os resultados possíveis de um **conectivo lógico** em função dos valores dos operandos.

Lógica Proposicional

Negação

p	$\neg p$
F	V
V	F

Por exemplo:

- p: O céu é azul
- seja $p = V$, $\neg p = F$
- Na linguagem C: !p

Lógica Proposicional

Disjunção

p	**q**	**p** $\vee$ **q**
F	F	F
V	F	V
F	V	V
V	V	V

Por exemplo:

- p: O céu é azul
- q: O céu é verde
- seja $p = V$ e $q = F$, $p \vee q = V$
- Na linguagem C: p || q

Lógica Proposicional

Conjunção

p	**q**	**p** $\wedge$ **q**
F	F	F
V	F	F
F	V	F
V	V	V

Por exemplo:

- p: O céu é azul
- q: O céu é verde
- seja $p = V$ e $q = F$, $p \wedge q = F$
- Na linguagem C: p && q

Lógica Proposicional

A lógica proposicional é importante para a programação pois permite modelar sentenças que influenciam no resultado de um programa

- p: O usuário clicou no botão x
- q: Os coeficientes que o usuário digitou resultaram em $\Delta < 0$
- r: O denominador da fração é 0
- s: O CPF digitado não é um CPF válido
- t: O número 7 não é divisível por 5
- u: O número 5 é primo

Operadores lógicos em C

Resultam em valores inteiros:

- *<expressão lógica>* && *<expressão lógica>*
- *<expressão lógica>* || *<expressão lógica>*
- ! *<expressão lógica>*

- Os valores lógicos em C são representados por inteiros: 0 representa falso e diferente de 0 verdadeiro (a não ser que use stdbool.h)
- Esses resultados estão de acordo com a **álgebra booleana**, na qual esses operadores são, respectivamente, $\wedge$, v e $\neg$.
- Esses operadores serão amplamente usados na elaboração dos algoritmos dessa disciplina.

Lógica Proposicional

Exercício em sala

Escreva um programa em C que declara em uma mesma linha dois números inteiros (identificador a e b).
O programa deve escrever na tela a mensagem **"Digite os valores de a e b: "** e ler do usuário os valores dos dois inteiros. Qual a **expressão lógica** que expressa o fato de um dos números dividir o outro? Em seguida, o programa deve escrever o valor lógico dessa expressão. Ou seja, vai escrever 1 (verdadeiro) se um dos números dividir o outro e 0 (falso) caso contrário.

Operadores relacionais

Operadores
Operadores relacionais

Operadores relacionais

Resultam em valores lógicos (1 ou 0)

- **==**: compara se os dois operandos são iguais
- **!=**: compara se os dois operandos são diferentes
- **>, <, >=, <=**: o mesmo significado matemático para $>, <, \geq, \leq$

Observações:

- Os operadores => ou =< não existem
- Por que $a > b > c$ não realiza o seu respectivo significado matemático?

Chamada de funções

Chamada de funções

- Em linhas gerais, as funções executam uma série de instruções que resultam em um **valor de retorno**

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

int main() {
    printf("Raiz de 15 = %f\n", sqrt(15.0));
    printf("3.5 elevado a 3 = %f\n", pow(3.5, 3));
    return 0;
}
```

- Os valores a serem colocados entre parênteses são os **argumentos**, um para cada **parâmetro** definido na função
- É necessário informar ao compilador gcc as **bibliotecas** para utilizar algumas funções específicas[4]
- Durante a compilação para algumas bibliotecas é necessário **linkar** (inclua -lm para math.h)

[4] um bom site para consulta: http://www.cplusplus.com/reference/

Exercício em sala

Escreva um programa em C que declara as seguintes variáveis:

- um **caractere** com valor 'g' (identificador simboloGravidade)
- um número **real** com valor 9.8196 (identificador gravidade)
- um número **real** com a altura (identificador altura)

O programa deve escrever na tela a mensagem **"Digite a altura da queda do objeto: "** e ler do usuário a altura. Em seguida, deve escrever na tela a seguinte mensagem:
O valor da gravidade g é: 9.8196 m/s2
O tempo de queda foi: ... segundos
Utilize o seguinte para calcular o tempo de queda:

$$\sqrt{\frac{2h}{g}}$$