

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

Introdução

Rafael Beserra Gomes

Universidade Federal do Rio Grande do Norte

Material compilado em 20 de fevereiro de 2017.

Licença desta apresentação:



<http://creativecommons.org/licenses/>

Linguagem 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

Linguagem 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

Linguagem 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 }
```

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

Linguagem 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

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, MS Visual Studio, etc

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

Variáveis e tipos

Representação dos dados na memória

Exemplo de dados na memória:

0xbfff22c	0	0	0	0	1	0	1	5 inteiro curto
0xbfff22d	0	1	0	0	0	1	0	B caractere
0xbfff22e	0	1	0	0	0	1	1	C caractere
0xbfff22f	1	1	0	1	1	1	0	3.2 real
0xbfff230	1	1	0	0	1	1	0	
0xbfff231	0	1	0	0	1	1	0	
0xbfff232	0	1	0	0	0	0	0	
0xbfff233	0	1	0	0	0	0	1	A caractere
0xbfff234	0	1	0	0	0	1	0	B caractere
0xbfff235	0	1	0	0	0	1	1	C caractere
0xbfff236	0	1	0	0	0	1	1	A caractere
0xbfff237	1	1	1	1	1	0	1	-5 inteiro curto
0xbfff238	0	0	0	0	0	0	0	1 inteiro
0xbfff239	0	0	0	0	0	0	0	
0xbfff23a	0	0	0	0	0	0	0	
0xbfff23b	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

¹a 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

Variáveis

- Experimente o seguinte código:

```
1 #include <stdio.h>
2
3 int main() {
4
5     int a = 3;
6     printf("Valor da variavel a: %d\n", a);
7     printf("Endereco da variavel a: %p\n", &a);
8     a = a + 2;
9     printf("Valor da variavel a: %d\n", a);
10
11     return 0;
12 }
```

Variáveis

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

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

Variáveis

Strings são sequências de caracteres

- uma string é representa entre aspas duplas

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

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

Variáveis

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

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

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

Variáveis

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)

³o 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);
```

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);
```

Exercício

Utilizaremos o **URI Online Judge** para alguns exercícios da disciplina.



- Crie sua conta
- Resolva o problema 1001 (categoria iniciante)
- Não esqueça de testar antes de enviar sua solução

Operadores

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

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

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>`

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

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

Lógica proposicional: exemplos de proposições

- 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 $p = V$, $\neg p = F$
- 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 $q = F$, $p \vee q = V$
- C: $p \parallel q$

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 $q = F$, $p \wedge q = F$
- C: `p && q`

Lógica proposicional: exemplos de proposições

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 , \vee e \neg .
- Esses operadores serão amplamente usados na elaboração dos algoritmos dessa disciplina.

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

Operador de chamada de funções

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

```
1 #include <math.h>
2 #include <stdio.h>
3
4 int main() {
5     printf("Raiz de 15 = %f\n", sqrt(15.0));
6     printf("3.5 elevado a 3 = %f\n", pow(3.5, 3));
7     return 0;
8 }
```

- 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 específicas⁴
- Durante a compilação para algumas bibliotecas é necessário **linkar** (inclua -lm para math.h)

⁴um bom site para consulta: <http://www.colusplus.com/reference/>

Exercício

URI Online Judge:

- 1002, 1005, 1012, 1013, 1016, 1018, 1020

Lista de Exercícios #1 e #2:

- Resolvam até 01/03/2017
- Dia 02/03/2017 correção de todos os exercícios e dúvidas