

# Linguagem C: variáveis, operadores, entrada/saída

Prof. Críston Algoritmos e Programação



# Linguagem C

- Linguagem de uso geral (qualquer tipo de aplicação)
  - Uma das linguagens mais utilizadas
  - Foi utilizada para desenvolver o sistema UNIX (e outros baseados neste, como Linux, Free-BSD)
- Disponível em vários sistemas operacionais e arquiteturas
  - Porém, algumas características podem depender da arquitetura (ex.: tamanho em bytes dos tipos de dados)
- Possui vários dialetos
  - Em 1988 surgiu a padronização "ANSI C", que incluiu também uma biblioteca de funções mínima.
  - Na biblioteca temos muitos algoritmos que podemos reutilizar!



## Linguagem C

- Relativamente "baixo nível"
  - Permite endereçar qualquer posição da memória
  - Não fornece operações para manipular diretamente textos, conjuntos, listas. Ex.: comparar textos.
  - Não fornece operações para entrada/saída
  - Mas podemos utilizar uma ampla bibliotecas de funções!
- Desvantagem: o programador comete mais erros que em linguagens de mais alto nível
  - Acesso indevido à memória.
  - Muitos erros não são avisados: assume que o programador sabe o que está fazendo
- Vantagens
  - Linguagem pequena (rápido aprendizado)
  - Eficiência (podemos codificar pensando em um problema específico)



## Linguagem C

- Compilada
  - Código-fonte (.c)código-objeto (.o)
  - Linkedição transforma vários objetos em um executável (.exe)
  - Assim, basta compilar código-objeto de código-fonte modificado
- Compilador gcc
  - #> man gcc
  - #> gedit primeiro.c &
  - #> gcc primeiro.c
  - #> ./a.out
  - hello!
  - #> \_
  - -ansi -Wall -c.. gcc primeiro.o



```
#include <stdio.h>
main()
{
    printf("Hello!\n");
}
```



- A linguagem C possui 32 palavras chaves.
- As palavras chaves não podem ser usadas para nenhum outro propósito.
- C diferencia palavras maiúsculas de minúsculas. Todas as palavras chaves devem ser escritas em minúsculo.
- Exempo:
  - if (A > B)
  - IF (a > b)



- Palavras reservadas (palavras-chave)
  - auto, break, case, char, const, continue,
  - default, do, double, else, enum, extern,
  - float, for, goto, if, int, long, register, return,
  - short, signed, sizeof, static, struct, switch,
  - typedef, union, unsigned, void, volatile, while.



- Todos os programas escritos em C consistem de uma ou mais funções.
- A função main() sempre precisa estar presente em um programa.
- A função main() é a primeira função a ser executada quando o programa for carregado.



Forma geral de um programa em C:

```
declaração de variáveis globais
tipo devolvido fn(lista de parâmetros)
{
    seqüência de comandos
}
tipo devolvido main(lista de parâmetros)
{
    seqüência de comandos
}
```



## Comentários em programas

- Comentar trechos do código fonte é uma forma de documentar o programa.
- A documentação serve para que os programas possam ser usados, corrigidos e aperfeiçoados pelo próprio programador ou por outras pessoas.
- Os comentário não influenciam em nada na execução do programa.



## Comentários em programas

 Em C comentários são cadeias de caracteres delimitadas pelas subcadeias
 /\* \*/

- Os comentários em C podem também ser de apenas uma linha, usando os caracteres //.
- Exemplo:

```
- /*Este programa ...*/- // Esta função faz ...
```



```
#include <stdio.h>

/*
Este programa altamente complexo escreve "Hello!" na tela.
*/
main()
{
    printf("Hello!\n"); // escreve Hello!
}
```



#### **Variáveis**

#### Nomes:

- Qualquer tamanho, mas apenas os 31 primeiros caracteres são considerados
- Não podemos utilizar palavras reservadas
- Podem ter letras ou dígitos
- Devem iniciar com letra ou '\_', mas devemos evitar começar com '\_', para n\u00e3o dar conflito com fun\u00fa\u00e3es da biblioteca
- Diferencia maiúsculas de minúsculas (ex.: 'x' e 'X' são variáveis diferentes)
- Prática tradicional:
  - quando a variável não modifica o valor durante o algoritmo (constante), utilize apenas letras maiúsculas
  - para as outras variáveis, utilize apenas minúsculas
  - deve sugerir a utilização: idade, altura, numero\_eleitores...



- Básicos:
  - Inteiros:
    - char 1 byte, permite armazenar 1 caracter ASCII
    - int tamanho depende da máquina hospedeira
    - short possivelmente menor que int
    - long possivelmente maior que int
  - Ponto flutuante (aproximação de números reais):
    - float precisão simples (quantidade de dígitos representativos)
    - double precisão dupla
- Qualificador unsigned indica que o número inteiro não tem sinal, assim ganhamos 1 bit na representação.
- Podemos utilizar sizeof(<tipo>) para determinar o número de bytes de cada tipo. Ex.: sizeof(char) retorna 1.



# Declaração de variáveis

- Variáveis devem ser declaradas antes de utilizadas
  - Em qualquer ponto do código, preferencialmente no início
- Formato: <tipo> <var1>, <var2>, ..., <varN>;- Ex.: int inicio, fim, incremento;
- Podemos inicializar uma variável na declaração
  - Ex.: int inicio = 0, fim, incremento = 1;
- Variáveis não inicializadas tem valor indefinido!
- Podemos utilizar o qualificador const para indicar que o valor da variável não pode ser modificado no código
  - A modificação pode gerar erro de compilação (depende do compilador)
  - Inicialização necessária
  - Ex.: const double PI = 3.14;



## Declaração de variáveis

- Tipos definidos pelo usuário
  - typedef char letra, digito;
  - typedef int tamanho;
  - typedef float preco;
  - Exemplo:
    - preco valor;
    - letra a, b, c;



## Comando de atribuição

- O comando de atribuição serve para guardar (atribuir) uma valor a uma variável.
  - Exemplo:
    - soma = A + B
    - fat = fatorial(x)



```
#include <stdio.h>
main()
{
    int i; // NÃO INICIALIZADA!!
    printf("%d\n", i);
}
```



```
#include <stdio.h>
main()
{
    const int i = 0;
    i = 1; // TENTATIVA DE MODIFICAR VARIÁVEL "const"
    printf("%d\n", i);
}
```



```
#include <stdio.h>
main()
    char c;
    int i;
    short s;
    long l;
    float f;
    double d;
    printf("%d\n", sizeof(double));
    printf("%d\n", sizeof(d));
}
```



```
    Tamanhos em nossas máquinas (inteiros em 32 bits)...

   - char(sempre 8 bits), short(16 bits), int(32 bits),
     long(igual ao int), float(32 bits), double(64 bits)
   - Maior char: 2^7 - 1 = 127 = 01111111
   - Menor char: -2^7 = -128 = 10000000
   - Maior unsigned char: 2^8 - 1 = 255 = 111111111
   - Menor unsigned char: 0 = 00000000
   - Maior short: 2^{15} - 1 = 32.767 = 01111111 111111111
   - Menor short: -2^{15} = -32.768 = 10000000 00000000
   - Maior unsigned short: 2<sup>16</sup>-1 =65.535= 11111111 11111111
   - Menor unsigned short:
```



O menor valor de cada tipo inteiro pode ser obtidos por overflow: o maior + 1. (representação em complemento de 2)

- Tamanhos em nossas máquinas (inteiros em 32 bits)...
  - char(sempre 8 bits), short(16 bits), int(32 bits),
    long(igual ao int), float(32 bits), double(64 bits)
  - Maior int:  $2^{31} 1 = 2.147.483.647$ 01111111 11111111 11111111 11111111
  - Menor int:  $-2^{31}$  = -2.147.483.64810000000 00000000 00000000 00000000
  - Maior unsigned int:  $2^{32}$  1 = 4.294.967.295 11111111 11111111 11111111 11111111
  - Menor unsigned int: 0
    00000000 00000000 00000000 00000000



```
#include <stdio.h>

main()
{
    char c = 128;
    unsigned char uc = 128;

    printf("%d\n", c);
    printf("%d\n", uc);
}
```



- Tamanhos em nossas máquinas...
  - char(sempre 8 bits), short(16 bits), int(32 bits),
    long(igual ao int), float(32 bits), double(64 bits)
  - Representação de ponto flutuante:
    - <sinal> 1.<mantissa> \* 2^<expoente> (se expoente <> 0)
    - <sinal> <mantissa nor> \* 2^<expoente> (se expoente = 0)
    - Expoente é somado com 2^<bits do expoente-1> 1
    - Mantissa normalizada: M.MMMMMMM
  - $Ex.: 1.0*2^{127} = 01111111 00000000 00000000 00000000$ 
    - 01111111(127) + 01111111(127) = 111111110



- Tamanhos em nossas máquinas...
  - float: 1bit(sinal) +8bits(exp) +23bits(mant) = 32bits
  - Expoente:  $-2^7 + 1$  até  $2^7 1 = -127$  até 127
    - 128 = Inf
  - Menor positivo:  $2^{-126-23} = 2^{-149} = 1.4*10^{-45}$ 
    - $2^{-126} = 00000000 10000000 00000000 00000000$
    - $2^{-127} = 00000000 01000000 00000000 00000000$
    - $2^{-128} = 00000000 00100000 00000000 00000000$
    - $2^{-149} = 00000000 00000000 00000000 00000001$
    - $2^{-150} = 0.0$  (underflow)
  - Maior: 01111111 01111111 11111111 11111111

    - 00010111(23) + 011111111(127) = 10010110
    - $(2^{24} 1)*2^{127-23} = 3.4*10^{38}$
  - 7 dígitos significativos:  $2^{24} = 16,777,216$ 
    - Cuidado:  $2^{24} + 1 = 2^{24}$  (erro de precisão)



```
    Tamanhos em nossas máquinas...

   - double: 1bit(sinal)+11bits(exp) +52bits(mant)= 64bits
    - Expoente: -2^{10} + 1 até 2^{10} - 1 = -1023 até 1023
        • 1024 = Inf
   - Menor positivo: 2^{-1022-52} = 2^{-1074} = 4.9*10^{-324}
        • 2^{-1022} = 00000000 00010000 \dots 00000000 00000000
        • 2^{-1923} = 00000000 00001000 \dots 00000000 00000000
        • 2^{-1924} = 00000000 00000100 \dots 00000000 00000000
        • 2^{-1074} = 00000000 000000000 \dots 00000000 00000001
        • 2^{-1075} = 0.0 (underflow)
    - Maior: 01111111 11101111 ... 11111111
        • 2^{53} - 1 = 01000011 00111111 \dots 111111111 11111111
        • 110100(52) + 111111111(1023) = 10000110011
        • (2^{53} - 1)*2^{1023-52} = 1.8*10^{308}
   - 15 dígitos significativos: 2^{53} = 9,007,199,254,740,992
```

• Cuidado:  $2^{53} + 1 = 2^{53}$  (erro de precisão)



```
#include <stdio.h>

main()
{
    float f1 = 16777216; // 2^24
    float f2 = 16777217; // 2^24 + 1
    double d = 16777217;

    printf("%f\n", f1);
    printf("%f\n", f2);
    printf("%f\n", d);
}
```





```
#include <stdio.h>
main()
{
    float f1 = 3.402823e+38; // 3.402823 * 10^38
    float f2 = 3.402824e+38; // 3.402824 * 10^38
    printf("%e\n", f1); // %e escreve em potência de 10
    printf("%e\n", f2);
}
```



#### **Constantes**

- Valores fixos que aparecem no código
- Todo número inteiro que cabe em um int é um int
  - Se não cabe, ele é um long
- Para torná-lo long, coloca-se um L (ou l) no final: ex.: 123L
- Zero antes do número inteiro indica valor em octal, 0X (ou 0x) antes do número indica valor hexadecimal
  - Ex.: 31 = 037 = 0x1F
- Para constantes unsigned, colocamos U (ou u) no final
  - Para unsigned long, colocamos UL (ou ul) no final
- Todo número com ponto decimal (12.3) ou expoente (1e-2) é considerado double
  - Para ser considerado float, coloca-se um F (ou f) no final



#### **Constantes**

- Para constantes char (1 caractere) utilizamos aspas simples, e é representada como o inteiro correspondente na tabela ASCII
  - Ex.: 'A' é representado como o inteiro 65, '0' como 48
  - Podemos realizar operações aritméticas com caracteres!
- Alguns caracteres especiais:
  - '\n' nova linha
  - '\t' tabulação
  - '\\' barra invertida
  - '\" aspas simples
  - '\" aspas duplas
  - '\0' armazena o número zero
  - '%%' representa o %



```
#include <stdio.h>
main()
{
    char c = 'A';
    printf("%d\n", c);
    printf("%c\n", c); // %c escreve o caracter (tabela ASCII)
}
```



## Conversões de tipo

- Implícitas (automáticas):
  - Expressões envolvendo mais de 1 tipo
    - Resulta no maior tipo. Ex.: int e long -> long, float e int -> float
  - Atribuição de expressão de um tipo em variável de outro
    - Informação pode ser perdida (truncagem) se o tipo da expressão for maior, mas a operação é permitida

```
Ex.: char c = 300; (alguns compiladores dão uma advertência)
```

- Explícita (forçada)
  - Formato: (<novo tipo>) expressão
  - Ex.: float x; sqrt((double) x);
  - Mesmo efeito da atribuição a uma variável do novo tipo
  - Utilidades:
    - Algumas funções exigem um tipo específico
    - Evita advertências e facilita verificação



## **Operadores aritméticos**

- Binários: + (soma) (subtração) \* (multiplicação) / (divisão) % (resto da divisão)
- Unários: (troca de sinal)
- Quando os operandos são inteiros, a divisão é inteira (a parte fracionária é truncada)
- O operador % n\u00e3o permite operandos float ou double



```
#include <stdio.h>
main()
{
    int a = 7, b = 2;
    printf("%d\n", a + b);
    printf("%d\n", a - b);
    printf("%d\n", a * b);
    printf("%d\n", a / b);
    printf("%f\n", ((double) a) / b);
    printf("%d\n", a % b);
    printf("%d\n", -a);
```



## Operadores relacionais e lógicos

- > >= < <= == (igual) != (diferente)
- && (e) || (ou) ! (não)
- Expressões são avaliadas das esquerda para a direita, até o resultado ser conhecido
  - Ou seja, parte da expressão pode não ser avaliada
  - Muitos algoritmos levam isso em conta (para garantir corretude)
- O número zero representa o valor FALSO, e qualquer número diferente de zero representa o VERDADEIRO
  - Expressões lógicas retornam 1 (verdadeiro) ou 0 (falso)
- Cuidado: é permitido fazer atribuições dentro de expressões. Erro comum: trocar igual (==) com atribuição (=)

```
- Ex.: x = 2; (x == 3); // vale 0 (FALSO)
 x = 2; (x = 3); // vale 3 (VERDADEIRO)
```

• O operador ! converte 0 em 1, e qualquer valor diferente de 0 em 0



```
#include <stdio.h>
main()
{
    int a = 7, b = 2;
    printf("%d\n", a == b);
    printf("%d\n", a != b);
    printf("%d\n", (a > 0) && (b <= 2));
    printf("%d\n", !a);
    printf("%d\n", a == 3);
    printf("%d\n", a = 3);
    printf("%d\n", a);
```



### Operadores de incremento e decremento

```
    ++ (soma 1 à variável) -- (subtrai 1 à variável)
    int x = 3;
    x++; // agora x vale 4
```

- Prefixado (++/-- antes da variável): utiliza na expressão o valor da variável depois do incremento
- Pós-fixado (++/-- depois da variável): utiliza na expressão o valor da variável antes do incremento

```
int x = 3, y;
y = x++; // agora x vale 4 e y vale 3
y = ++x; // agora x e y valem 5
```



### **Operadores bit-a-bit**

& (e) | (ou) ^ (ou exclusivo) ~ (complemento)
 << (deslocamento a esquerda) >> (deslocamento a direita)



# Operadores de atribuição

• = (atribuição simples)

```
int x = 2;
x = 2*3 + 1 // x vale 7
x += 2; // x vale 9 (o mesmo que <math>x = x + 2)
x -= 5; // x vale 4 (o mesmo que x = x - 5)
x *= 2; // x vale 8 (o mesmo que <math>x = x * 2)
x /= 3; // x vale 2 (o mesmo que <math>x = x / 3)
x = 3; // x \text{ vale 2 (o mesmo que } x = x % 3)
x <<= 1; // x vale 4 (o mesmo que x = x << 1)
x >>= 1; // x vale 1 (o mesmo que x = x << 2)
x \&= 3; // x vale 1 (o mesmo que x = x \& 3)
x = 4; // x vale 5 (o mesmo que x = x + 4)
x ^= 7; // x vale 2 (o mesmo que x = x ^7)
```

• += -= \*= /= %= <<= >>= &= |= ^=



### Precedência de operadores

```
++ -- +(unário) -(unário) (<tipo>) sizeof
  !=
&
&&
           /= %= <<= >>= &= |=
```



#### Saída formatada

```
printf("<formato>", arg1, arg2, ..., argN);
```

- Converte os argumentos em caractere de acordo com <formato> e imprime na tela
- <formato>:
  - Caracteres comuns são mantidos
  - O n-ésimo %<caractere de conversão> é associado ao n-ésimo argumento

```
int i = 1;
double media = 7.0;
printf("resultado do aluno %d = %f\n", i, media);
```



#### Saída formatada

- Caracteres de conversão
  - d, i: int, número decimal
  - o: int, número octal
  - x: int, número hexadecimal
  - u: int, sem sinal
  - c: int, 1 caractere
  - s: cadeia de caracteres (veremos posteriormente)
  - f: double, [-]m.dddddd (default 6 casas decimais)
  - e: double, no formato x.x e +y (ex.: 1.2e+3 == 1200)
  - g: double, formato do 'f' se expoente menor que –4 ou maior que o número de casas decimais, formato do 'e' caso contrário
  - %: caractere '%'



```
int x = 31;
printf("%d\n", x); // 31
printf("%o\n", x); // 37
printf("%x\n", x); // 1f
x = -1;
printf("%u\n", x); // 4294967295
x = 64;
printf("%c\n", x); // A
double d = 1200.0;
printf("%f\n", d); // 1200.000000
printf("%e\n", x); // 1.200000e+003
printf("%g\n", x); // 1200.000000
```



### Saída formatada

- Entre o % e o caractere de conversão podemos ter
  - Número indicando a largura mínima do campo

```
double d = 1.0;
printf("%10f\n", d); // [ 1.000000]
```

- Um ponto separando a largura do número de casas decimais

```
printf("%10.2f\n", d); // [ 1.00]
```

Sinal de menos, indicando alinhamento a esquerda

```
printf("%-10f\n", d); // [1.000000 ]
```



#### Entrada formatada

```
scanf("<formato>", &var1, &var2, ..., varN);
```

 Lê da entrada padrão (teclado) uma sequência de caracteres, interpreta de acordo com <formato>, e grava os valores obtidos nas variáveis (na ordem encontrada)

```
int i, r; float d;
r = scanf("%d, %f", &i, &d);
// espera inteiro e real separados p/ vírgula
```

- Espaços e tabulação são ignorados
- Outros caracteres devem casar exatamente
- Retorna número de campos casados
- Caracteres de conversão: use 'lf' para double



- Pré-processamento é uma fase de compilação anterior à tradução do programa para a linguagem assembly.
- O pré-processamento engloba, entre outras, tarefas tais como a inclusão de arquivos auxiliares, a tradução de constantes simbólicas e de macros.
- As diretivas de pré-processamento são comandos auxiliares que sinalizam ao compilador a necessidade de realizar tarefas de préprocessamento.



- Na linguagem C o caractere # é a diretiva que indica a necessidade de pré-processamento.
- O compilador pode incluir arquivos quando isto for indicado no conteúdo de outro aquivo.
- Exemplo:
  - #include <stdio.h>
  - #include <sistemas.c>



- A linguagem C possui a diretiva #define que serve para incluir no programa constantes simbólicas e expressões denominadas macros.
- Exemplo:
  - #define PI 3.14159
  - #define se if
  - #define eq =



- Macros são expressões usadas para abreviar construções muito longas, usadas como muita freqüência num programa.
- Exemplo:
  - #define EHPAR(X) (((x)%2)?0:1)
  - #define ERRO(mens) printf("ERRO: %s\n",mens)