Variáveis e Tipos

Programação e Desenvolvimento de Software I

Gleison S. D. Mendonça, Luigi D. C. Soares
{gleison.mendonca, luigi.domenico}@dcc.ufmg.br

Notação Decimal vs Binária

- Decompondo um número representado na notação decimal

$$19.625 =$$

Notação Decimal vs Binária

- Decompondo um número representado na notação decimal

$$19.625 = 1 \times 10^1 + 9 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$$

Notação Decimal vs Binária

- Decompondo um número representado na notação decimal

$$
\begin{aligned}
19.625 &= 1 \times 10^1 + 9 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3} \\
&= 10 + 9 + 0.6 + 0.02 + 0.005
\end{aligned}
$$

Notação Decimal vs Binária

- De binário para decimal

$$10011.101 =$$

Notação Decimal vs Binária

- De binário para decimal

$$10011.101 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

Notação Decimal vs Binária

- De binário para decimal

$$
\begin{aligned}
10011.101 &= 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + \\
&\quad 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\
&= 16 + 0 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\
&= 19.625
\end{aligned}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$19 \;/\; 2 =$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 19 \,/\, 2 = \\
1 & \quad 9 \,/\, 2 = \\
\end{array}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{aligned}
19 \;/\; 2 &= \\
1 \quad 9 \;/\; 2 &= \\
1 \quad 4 \;/\; 2 &=
\end{aligned}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 19 \,/\, 2 = \\
1 & \quad 9 \,/\, 2 = \\
1 & \quad 4 \,/\, 2 = \\
0 & \quad 2 \,/\, 2 =
\end{array}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 19 \,/\, 2 = \\
1 & \quad 9 \,/\, 2 = \\
1 & \quad 4 \,/\, 2 = \\
0 & \quad 2 \,/\, 2 = \\
0 & \quad 1 \,/\, 2 =
\end{array}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 19 \,/\, 2 = \\
1 & \quad 9 \,/\, 2 = \\
1 & \quad 4 \,/\, 2 = \\
0 & \quad 2 \,/\, 2 = \\
0 & \quad 1 \,/\, 2 = \\
1 & \quad 0
\end{array}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 19 \,/\, 2 = \\
\uparrow\, 1 & 9 \,/\, 2 = \\
1 & 4 \,/\, 2 = \\
0 & 2 \,/\, 2 = \\
0 & 1 \,/\, 2 = \\
1 & 0
\end{array}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$0.625 \times 2 =$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{aligned}
0.625 &\times 2 = \\
1 + \quad 0.25 &\times 2 =
\end{aligned}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{aligned}
& 0.625 \times 2 = \\
1 + \quad & 0.25 \times 2 = \\
0 + \quad & 0.5 \times 2 =
\end{aligned}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{aligned}
& 0.625 \times 2 = \\
1 + \quad & 0.25 \times 2 = \\
0 + \quad & 0.5 \times 2 = \\
1 + \quad & 0
\end{aligned}
$$

Notação Decimal vs Binária

- Caminho inverso: decimal para binário

$$
\begin{array}{rl}
 & 0.625 \times 2 = \\
1 + & 0.25 \times 2 = \\
0 + & 0.5 \times 2 = \\
1 + & 0
\end{array}
$$

Notação Decimal vs Binária

- $0.6 = ?$
- Quantos bits precisamos depois do “.”?

Notação Decimal vs Binária

- $0.6 = 0.10011001\ldots$
- Quantos bits precisamos depois do "."? **Infinitos**

$$
\begin{aligned}
& 0.6 \times 2 = \\
1 + \quad & 0.2 \times 2 = \\
0 + \quad & 0.4 \times 2 = \\
0 + \quad & 0.8 \times 2 = \\
1 + \quad & 0.6 \times 2 = \ldots
\end{aligned}
$$

Representando números inteiros

- Forma mais simples: **sinal-magnitude**
  - O **bit mais significativo** corresponde ao sinal
  - Os demais correspondem ao valor absoluto do número

Representando números inteiros

- Forma mais simples: **sinal-magnitude**
  - O **bit mais significativo** corresponde ao sinal
  - Os demais correspondem ao valor absoluto do número
- Exemplo: considere uma representação usando cinco dígitos binários (**bits**)

Decimal	Binário
+5	**0** 0101
−3	**1** 0011

```
    00101 (+5)
+   10011 (-3)
-----------
    11000 (-8 ???)
```

Representando números inteiros

- Forma mais simples: **sinal-magnitude**
  - O **bit mais significativo** corresponde ao sinal
  - Os demais correspondem ao valor absoluto do número
- **Desvantagens**:
  - A representação dificulta os cálculos
  - Duas notações para o zero ($+0$ e -0)

Representando números inteiros

- Forma mais utilizada: **complemento de 2**
  - Números positivos: idêntica à forma sinal-magnitude

Representando números inteiros

- Forma mais utilizada: **complemento de 2**
  - Números positivos: idêntica à forma sinal-magnitude
  - Números negativos: a representação se dá em dois passos
    1. Inverter os bits (0 vira 1, 1 vira 0) da representação do número positvo
    2. Somar 1 ao resultado

Representando números inteiros

- Forma mais utilizada: **complemento de 2**
  - Números positivos: idêntica à forma sinal-magnitude
  - Números negativos: a representação se dá em dois passos
    1. Inverter os bits (0 vira 1, 1 vira 0) da representação do número positvo
    2. Somar 1 ao resultado

Decimal	Binário
$+6$	**0** 0110
-6	**1** 1010
$+5$	**0** 0101
-3	**1** 1101

$$\begin{array}{rr} & 00101\ (+5) \\ + & 11101\ (-3) \\ \hline & 00010\ (+2) \end{array}$$

Representando números reais

- Representação com ponto fixo: $12,34$
- Representação com **ponto (vírgula) flutuante**: $0,1234 \times 10^2$
- A representação com ponto flutuante segue padrões internacionais (IEEE-754 e IEC-559)

Representando dados não-numéricos

- Padrões internacionais para a codificação de caracteres (**ASCII**, **ANSI**, **Unicode**).
- A Linguagem C adota o padrão ASCII (American Standard Code for Information Interchange):
  - Código para representar caracteres como números
  - Cada caractere é representado por **1 byte**, ou seja, uma **seqüência de 8 bits**
- `https://pt.wikipedia.org/wiki/ASCII`

Representando dados não-numéricos

- Padrões internacionais para a codificação de caracteres (**ASCII**, **ANSI**, **Unicode**).
- A Linguagem C adota o padrão ASCII (American Standard Code for Information Interchange):
  - Código para representar caracteres como números
  - Cada caractere é representado por **1 byte**, ou seja, uma **seqüência de 8 bits**
- `https://pt.wikipedia.org/wiki/ASCII`

Caractere	Decimal	Binário
\n	10	0000 1010
0	48	0011 0000
@	64	0100 0000
A	65	0100 0001
a	97	0110 0001

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar
- Função principal (**main**): ponto de entrada do programa

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar
- Função principal (**main**): ponto de entrada do programa
- A linguagem C é **case sensitive**
  - **Main** ou **MAIN**, por exemplo, provocam erros de sintaxe

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar
- Função principal (**main**): ponto de entrada do programa
- A linguagem C é **case sensitive**
  - **Main** ou **MAIN**, por exemplo, provocam erros de sintaxe
- Escrever um programa em C corresponde a escrever o **corpo** da função principal

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar
- Função principal (**main**): ponto de entrada do programa
- A linguagem C é **case sensitive**
  - **Main** ou **MAIN**, por exemplo, provocam erros de sintaxe
- Escrever um programa em C corresponde a escrever o **corpo** da função principal
- O **corpo** de uma função sempre começa com abre-chaves **{** e termina com fecha-chaves **}**

Escrevendo um programa em C

- Inclusão dos cabeçalhos das bibliotecas que vamos utilizar
- Função principal (**main**): ponto de entrada do programa
- A linguagem C é **case sensitive**
  - **Main** ou **MAIN**, por exemplo, provocam erros de sintaxe
- Escrever um programa em C corresponde a escrever o **corpo** da função principal
- O **corpo** de uma função sempre começa com abre-chaves **{** e termina com fecha-chaves **}**
- Comandos devem terminar com ponto e vírgula **;**

Escrevendo um programa em C

```c
#include <stdio.h> // Cabeçalho da biblioteca de entrada/saída
// Função main: ponto de entrada do programa ("//" indica um comentário)
int main(int argc, char *argv[]) {
    /*
      Corpo da função principal
      (comentário em múltiplas linhas)
     */
    return 0;
}
```

Variáveis

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int a = 7;
    int b = 3;
    int q = 0; // Inicializando quociente
    while (b <= a) {
        q = q + 1; // Somar 1 ao valor de q
        a = a - b; // Subtrair B do valor de A
    }
    // ...
    return 0;
}
```

Variáveis

- Os dados de um programa precisam ser armazenados na **memória** do computador

Variáveis

- Os dados de um programa precisam ser armazenados na **memória** do computador
- Cada posição de memória possui um **endereço**

Variáveis

- Os dados de um programa precisam ser armazenados na **memória** do computador
- Cada posição de memória possui um **endereço**
- Uma variável é um **nome simbólico** (ou etiqueta) de uma posição de memória

Variáveis

- Os dados de um programa precisam ser armazenados na **memória** do computador
- Cada posição de memória possui um **endereço**
- Uma variável é um **nome simbólico** (ou etiqueta) de uma posição de memória

Variável	Endereço	...	b_{24}	b_{25}	b_{26}	b_{27}	b_{28}	b_{29}	b_{30}	b_{31}
a	e_0	0	0	0	0	0	0	1	1	1
b	e_1	0	0	0	0	0	0	0	1	1
q	e_2	0	0	0	0	0	0	0	0	0

Variáveis

- Os dados de um programa precisam ser armazenados na **memória** do computador
- Cada posição de memória possui um **endereço**
- Uma variável é um **nome simbólico** (ou etiqueta) de uma posição de memória
- Seu **conteúdo pode variar** durante a execução do programa

Variável	Endereço	...	b_{24}	b_{25}	b_{26}	b_{27}	b_{28}	b_{29}	b_{30}	b_{31}
a	e_0	0	0	0	0	0	0	1	**0**	**0**
b	e_1	0	0	0	0	0	0	0	1	1
q	e_2	0	0	0	0	0	0	0	0	**1**

Variáveis

- Deve ser definida antes de ser utilizada:

Variáveis

- Deve ser definida antes de ser utilizada:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // Error: 'X' undeclared (first use in this function)
    printf("X: %d\n", X);
    return 0;
}
```

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x, y;
    x = 10;
    y = 20;
    printf("X: %d\n", x);
    printf("Y: %d\n", y);
    return 0;
}
```

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    /* Error: 'X' undeclared (first use in this function)
       Note que 'X' é diferente de 'x' */
    printf("X: %d\n", X);
    return 0;
}
```

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // Error: expected identifier or '(' before numeric constant
    int 1;
    return 0;
}
```

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)
  - Caracteres devem ser letras, números ou underscores

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)
  - Caracteres devem ser letras, números ou underscores

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // Error: expected '=', ',', ';', 'asm' or
    // '__attribute__' before numeric constant
    int teste.123;
    return 0;
}
```

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)
  - Caracteres devem ser letras, números ou underscores
  - Palavras-chave não podem ser usadas como nomes

Variáveis

- Deve ser definida antes de ser utilizada: **tipo_variável lista_de_variáveis**
- Nome:
  - *Case sensitive* (letras maiúsculas e minúsculas são consideradas diferentes)
  - Pode ter um ou mais caracteres
  - Deve iniciar com letras ou underscore (_)
  - Caracteres devem ser letras, números ou underscores
  - Palavras-chave não podem ser usadas como nomes

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // Error: expected identifier or '(' before 'while'
    int while;
    return 0;
}
```

Variáveis

- Lista de 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 typeof union unsigned void volatile while**

Variáveis

- Tipo: Define os valores que ela pode assumir e as operações que podem ser realizadas com ela
- Exemplo:
  - Tipo **int** recebe apenas valores inteiros
  - Tipo **float** armazena apenas valores reais

Tipos

- Diferentes tipos possuem tamanhos diferentes (em **bytes**, 1 byte = 8 bits)
- Tamanhos podem variar de acordo com o sistema

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Tamanho char: %d bytes\n", sizeof(char));
    printf("Tamanho short: %d bytes\n", sizeof(short));
    printf("Tamanho int: %d bytes\n", sizeof(int));
    printf("Tamanho long: %d bytes\n", sizeof(long));
    printf("Tamanho float: %d bytes\n", sizeof(float));
    printf("Tamanho double: %d bytes\n", sizeof(double));
    return 0;
}
```

Tipos

- Diferentes tipos possuem tamanhos diferentes (em **bytes**, 1 byte = 8 bits)
- Tamanhos podem variar de acordo com o sistema

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Tamanho long double: %d bytes\n", sizeof(long double));
    printf("Tamanho unsigned char: %d bytes\n", sizeof(unsigned char));
    printf("Tamanho unsigned short: %d bytes\n", sizeof(unsigned short));
    printf("Tamanho unsigned int: %d bytes\n", sizeof(unsigned int));
    printf("Tamanho unsigned long: %d bytes\n", sizeof(unsigned long));
    return 0;
}
```

Tipos

- Diferentes tipos possuem tamanhos diferentes (em **bytes**, 1 byte = 8 bits)
- Tamanhos podem variar de acordo com o sistema

```c
#include <stdio.h>
#include <limits.h>
int main(int argc, char *argv[]) {
    printf("char: %d a %d\n", CHAR_MIN, CHAR_MAX);
    printf("short: %d a %d\n", SHRT_MIN, SHRT_MAX);
    printf("int: %d a %d\n", INT_MIN, INT_MAX);
    printf("long: %ld a %ld\n", LONG_MIN, LONG_MAX);
    return 0;
}
```

Tipos

- Diferentes tipos possuem tamanhos diferentes (em **bytes**, 1 byte = 8 bits)
- Tamanhos podem variar de acordo com o sistema

```c
#include <stdio.h>
#include <limits.h>
#include <float.h>
int main(int argc, char *argv[]) {
    printf("unsigned char: %u a %u\n", 0, (unsigned int) UCHAR_MAX);
    printf("float: %e a %e\n", -FLT_MAX, FLT_MAX);
    printf("double: %e a %e\n", -DBL_MAX, DBL_MAX);
    return 0;
}
```

Atribuição

- Operador de atribuição: =
- **nome_da_variável** = **expressão, valor ou constante;**
- A linguagem C suporta múltiplas atribuições

Atribuição

- Operador de atribuição: =
- nome_da_variável = expressão, valor ou constante;
- A linguagem C suporta múltiplas atribuições

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 5;
    int y, z;
    z = y = x + 3;
    char c = 'a'; // Caracteres ficam sempre entre aspas simples
    return 0;
}
```

Conversão de tipos

- O compilador converte automaticamente o valor do lado direito para o tipo do lado esquerdo da atribuição
- A conversão também pode ser feita de forma explícita
- **Pode haver perda de informação**

Conversão de tipos

- O compilador converte automaticamente o valor do lado direito para o tipo do lado esquerdo da atribuição
- A conversão também pode ser feita de forma explícita
- **Pode haver perda de informação**

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    float x = 10.5;
    int y = x; // y recebe somente a parte inteira de x
    int z = (int) x; // Convertendo de forma explícita
    return 0;
}
```

Constantes

- Como uma variável, uma constante também armazena um valor na memória do computador
- Entretanto, esse valor não pode ser alterado
- Para constantes é obrigatória a atribuição do valor

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    const double pi = 3.1415;
    return 0;
}
```

Constantes char

Código	Comando
\b	retrocesso (backspace)
\n	nova linha (new line)
\t	tabulação horizontal
\'	Apóstrofo
\"	Aspas
\\	Barra invertida (backslash)
...	...

Comando de saída

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Inteiro: %d\n", 10);
    printf("Real: %f\n", 10.5);
    printf("Inteiro (long): %ld\n", 10 + 2);
    printf("Múltiplos valores: %d \t vs \t %f\n", 1, 3.14);
    printf("Caractere: %c\n", 'a');
    printf("String (sequência de caracteres): %s\n", "hello world");
    return 0;
}
```

Comando de saída

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    // Inteiros com zeros a esquerda:
    printf("%08d\n", 10);
    // Formatação de casas decimais:
    printf("%.3f\n", 3.1415);
    // Notação científica:
    printf("%e\n", 32.1415);
    return 0;
}
```

Comando de entrada

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x;
    scanf("%d", &x); // Leitura de um único valor
    printf("x: %d\n", x);

    float y;
    scanf("%d%f", &x, &y); // Leitura de múltiplos valores
    printf("x: %d\t y: %f\n", x, y);

    return 0;
}
```

Referências / Agradecimentos

- Linguagem C completa e descomplicada, André Backes
- Material adaptado:
  - Prof. Fabrício Benevenuto (`https://homepages.dcc.ufmg.br/~fabricio/`)
  - Prof. Pedro O. S. Vaz de Melo (`https://homepages.dcc.ufmg.br/~olmo/`)