Algoritmos e Programação

Aula 13

Cálculos em C

Biblioteca <math.h>

Conteúdo

- Cálculos em C

Biblioteca matemática

- Em C, podemos calcular o logaritmo neperiano de um número x utilizando a seguinte série infinita:

$$\ln(x) = \left(\frac{x-1}{x}\right)^1 + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 + \frac{1}{4}\left(\frac{x-1}{x}\right)^4 + \cdots$$

Biblioteca matemática

- Em C, podemos calcular o logaritmo neperiano de um número x utilizando a seguinte série infinita:

$$\ln(x) = \left(\frac{x-1}{x}\right)^1 + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 + \frac{1}{4}\left(\frac{x-1}{x}\right)^4 + \cdots$$

- Ou então, podemos utilizar uma **biblioteca matemática** que implemente a função logaritmo neperiano, e desta maneira, não precisamos nos preocupar com este cálculo!
- C oferece a biblioteca **`math.h`** que implementa diversas funções matemáticas.

Biblioteca **`math.h`**

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

- Constantes: **`M_PI`** $= \pi = 3.141592$, **`M_E`** $= e = 2.718281$
- **`fabs(x)`** = calcula $|X|$, o valor absoluto (módulo) de x
- **`exp(x)`** = calcula e^x
- **`log(x)`**, **`log10(x)`** calculam $\log_e(x)$ e $\log_{10}(x)$
- **`sin(x)`**, **`cos(x)`**, **`tan(x)`** = calculam seno, cosseno e tangente
- **`sqrt(x)`**, **`pow(x,y)`** calculam $\sqrt{x}$ e x^y, respectivamente
- **`ceil(x)`**, **`floor(x)`**, **`round(x)`** = calculam $\lfloor x \rfloor$(menor inteiro), $\lceil x \rceil$(maior inteiro) e o inteiro mais próximo de x, respectivamente.

Cuidado na manipulação de números

- A linguagem C possui uma série de **comportamentos pré-definidos** em relação a **cálculos aritméticos**, que podem afetar a precisão e correção dos resultados.

Cuidado na manipulação de números

- A linguagem C possui uma série de **comportamentos pré-definidos** em relação a **cálculos aritméticos**, que podem afetar a precisão e correção dos resultados.
- A seguir vamos comentar sobre alguns cuidados que devemos ter ao fazer cálculos matemáticos:
  - Tamanho dos dados e faixa de representação
  - Conversão de tipos
  - Divisão inteira e fracionária
  - Zeros à esquerda dos números

Tamanho dos tipos de dados

Como C é portável, nosso código pode ser compilado para máquinas com tamanhos diferentes de registradores (16 bits, 32 bits, 64 bits ...)

Tamanho dos tipos de dados

Como C é portável, nosso código pode ser compilado para máquinas com tamanhos diferentes de registradores (16 bits, 32 bits, 64 bits ...)

O padrão ANSI C estabelece tamanhos mínimos para os dados:

Tipo	Tamanho mínimo (bits)	Faixa mínima de valores
char	8	com sinal: -128 a 127 sem sinal: 0 a 255
int	16	com sinal: -32768 a 32767 sem sinal: 0 a 65535
float	32	+/- 3.4e +/- 38
double	64	+/- 1.7e +/- 308

Contudo, o tamanho real usado depende da máquina. Processadores atuais usam valores de palavras maiores que os mínimos.

Operador `sizeof`

Podemos utilizar o comando **`sizeof`** para descobrir o tamanho real de uma variável ou de um tipo de dado, **em bytes**.

Operador `sizeof`

Podemos utilizar o comando **`sizeof`** para descobrir o tamanho real de uma variável ou de um tipo de dado, **em bytes**.

```c
#include <stdio.h>

int main() {
    char c;
    printf("Um int ocupa %d byte(s), float ocupa %d
  byte(s).\n", sizeof(int), sizeof(float));
    printf("A variavel c ocupa %d byte(s).\n", sizeof(c));
}
```

Saída (hipotética):

```
Um int ocupa 4 byte(s), float ocupa 4 byte(s).
A variável c ocupa 1 byte(s).
```

Modificadores de tipos

C provê **modificadores** para alterar certas características dos tipos:

Tamanho:

- **`long`** – solicita aumento no tamanho da variável
- **`short`** – solicita redução no tamanho da variável

Modificadores de tipos

C provê **modificadores** para alterar certas características dos tipos:

Tamanho:

- **`long`** – solicita aumento no tamanho da variável
- **`short`** – solicita redução no tamanho da variável

```c
#include <stdio.h>
int main() {
    long int x = 3000000000;
    short int y = 121;
    printf("%ld\n", x);
    printf("%hd\n", y);
}
```

Saída (hipotética):

```
-1294967296
121
```

Modificadores de tipos

Sinal (tipos inteiros):

- **`signed`** – converte tipo sem sinal (char) em com sinal.
- **`unsigned`** – converte tipo com sinal (**`int, long int, short int`**) em sinal.

Modificadores de tipos

Sinal (tipos inteiros):

- **`signed`** – converte tipo sem sinal (char) em com sinal.
- **`unsigned`** – converte tipo com sinal (**`int, long int, short int`**) em sinal.

```c
#include <stdio.h>
int main() {
    unsigned long int x = 3000000000;
    signed char b = -13;
    printf("%lu\n", x);
    printf("%d\n", b);
}
```

Saída (hipotética):

```
3000000000
-13
```

Operações com números fracionários

Em C, a operação com números fracionários possui algumas particularidades.

Ex:

```c
#include <stdio.h>
int main() {
    int i;
    float f = 6.6;
    i = (f / 2) / 2.2;
    f = i / 10;
    printf("i = %d\nf = %f\n", i, f);
}
```

Saída:

```
i = 1
f = 0.000000
```

Conversão de tipos explícita

Em C, podemos *forçar* a **conversão** de um tipo **fracionário em inteiro**, e vice-versa.

Basta fixar o valor desejado com o tipo que queremos converter:

Conversão de tipos explícita

Em C, podemos *forçar* a **conversão** de um tipo **fracionário em inteiro**, e vice-versa.

Basta fixar o valor desejado com o tipo que queremos converter:

```c
#include <stdio.h>
int main() {
    int i;
    float f = 6.6;
    i = (int) (f / 2) / 2.2;
    f = (float) i / 10;
    printf("i = %d\nf = %f\n", i, f);
}
```

Saída:

```
i = 1
f = 0.100000
```

Conversão de tipos implícita

Regras de conversão:

- ao atribuir o conteúdo de um **`int`** para um **`double/float`**, ele é convertido para representação em ponto flutuante (real).
- Ao atribuir o conteúdo de um **`double/float`** para um **`int`**, toda a parte após a vírgula é **truncada** (eliminada)

Conversão de tipos implícita

Regras de conversão:

- ao atribuir o conteúdo de um **`int`** para um **`double/float`**, ele é convertido para representação em ponto flutuante (real).
- Ao atribuir o conteúdo de um **`double/float`** para um **`int`**, toda a parte após a vírgula é **truncada** (eliminada)

```
#include <stdio.h>
int main() {
    int i = 12;
    int j = 6;
    float f = 13.8;
    i = f;
    f = j;
    printf("i = %d\nf = %f\n", i, f);
}
```

Saída:

i = 13

f = 6.000000

Divisão inteira e fracionária

O mesmo operador / executa divisões inteiras e fracionárias.

Regra: se ambos os operandos são inteiros, a divisão é inteira. Caso contrário, a divisão é fracionária. Exemplo:

Divisão inteira e fracionária

O mesmo operador / executa divisões inteiras e fracionárias.

Regra: se ambos os operandos são inteiros, a divisão é inteira. Caso contrário, a divisão é fracionária. Exemplo:

```
#include <stdio.h>
int main() {
    int i;
    float f1,f2;
    i = 3 / 2;
    f1 = 3 / 2;
    f2 = 3.0 / 2;
    printf("%d, %f, %f", i, f1, f2);
}
```

Saída: 1, 1.000000, 1.500000

Dica: para forçar a divisão fracionária de números exatos como o 3, adicione um .0, ou faça uma *conversão explícita de tipos*.

Números precedidos por 0

- Valores numéricos em C podem ser descritas em forma hexadecimal (base 16) e octal (base 8), além de decimal.

Usam-se prefixos para indicar a base de um número:

- **0x** = **hexadecimal**: 0x29A (= 666 decimal)
- **0** = **octal**: 015 (= 13 decimal)

Cuidado, não coloque zeros à esquerda dos números em C:

$$015 \neq 15$$