Arquitetura de Computadores

MIEI – 2017/18
DI-FCT/UNL
Aula 5

Lendo números em C

- Ler *standard input* convertendo à representação da arquitetura:

```
int scanf( char *format, …vars…)
```

  - Devolve nº de variáveis lidas
- Exemplo:

```
int v1, v2;
int n = scanf("%d%d",&v1,&v2);
if (n!=2) ... // erro
```


Lendo números em C

- Exemplo do "ascii to int":

```
int atoi( char *str );
```

- ler uma sequência de inteiros de argv:

```
int main( int argc, char *argv[] ) {
   int nums[100];
   if (argc>100 || argc==1) ... // erro
   for (int i=1; i<argc; i++)
      nums[i-1] = atoi( argv[i] );
...
```


Representação de números reais

- Os inteiros têm uma gama limitada de valores:
  - não podem representar números fraccionários
  - números grandes necessitam de muitos bits
- Pretende-se uma grande gama de valores reais, desde muito grandes a muito pequenos.
- Exemplos:
  - massa do Sol: 2×10^{33} gr.
  - massa do electrão: 9×10^{-28} gr.

    intervalo de valores → 10^{61}

Representação de números reais

- Número de dígitos significativos necessários:
  - massa do electrão: 9×10^{-28} g.

    0 , 0--------9
  - massa do Sol: 2×10^{33} g.

    2--------0 , 0

    34 dígitos , 28 dígitos → **62 dígitos significativos em base 10! e em binário?**
- Conclusão: tal é um desperdício
  - para quê tanta precisão para a massa do Sol?

Notação Científica

- Pretende-se representar:
  - A gama de valores a representar (a escala) e
  - O número de dígitos significativos (a precisão)
- A notação científica permite isso:
  - na base 10: $m \times 10^e$
  - gama de valores → o número de dígitos do expoente *e*
  - precisão → o número de dígitos da mantissa *m*
- Exemplo:
  - massa do Sol: 2×10^{33} g.
  - massa do electrão: 9×10^{-28} g.

Representação de números reais

- Representações possíveis em 9 dígitos:
  - virgula fixa

    exemplo (9 dígitos c/3 para parte fracionária):

    032767,004
  - virgula flutuante (ou notação científica)

    exemplo (8 dígitos c/ mais um para escala)

    $3{,}2767004 \times 10^4$

    ou os exemplos anteriores:

    massa do Sol: 2×10^{33} gr.

    massa do electrão: 9×10^{-28} gr.

Reais em binário

- Valor de cada digito em base dez:
  - $23{,}45 = 2x10^1+3x10^0+4x10^{-1}+5x10^{-2}$

    10 1 1/10=0,1 1/100=0,01
- Valor de cada digito em binário:
  - $d_1d_0,d_{-1}d_{-2} = d_1x2^1+d_0x2^0+d_{-1}x2^{-1}+d_{-2}x2^{-2}$

    2 1 ½=0,5 ¼=0,25
- Exemplos:
  - $1{,}5_{10} = 1{,}1_2 \rightarrow 1x2^0+1x2^{-1}$
  - $23{,}125 = 10111{,}001_2$

Para a parte decimal 0,125:
0,125 x 2 = **0**,25
0,25 x 2 = **0**,5
0,5 x 2 = **1**,0

Representação de Números Reais

- Os computadores modernos permitem representar números reais no formato de vírgula flutuante
  - em base 2 será: $m \times 2^e$
  - há que codificar m e e
- Possuem *hardware* especializado para operar com este tipo de dados:
  - *FPU - Floating Point Unit*

Representação em vírgula flutuante

- Composta por três partes:
  - Sinal
  - Expoente
  - Mantissa
- Exemplo de possíveis representações:
  - Suponhamos o número 17,5

Em binário $17{,}5_{10}$ é $10001{,}1_2 \times 2^0 =$

$= 1000{,}11_2 \times 2^1 = 100{,}011_2 \times 2^2 = 10{,}0011_2 \times 2^3 =$

$= 1{,}00011_2 \times 2^4 = 0{,}100011_2 \times 2^5$

Sinal, expoente, mantissa

- Uma possível codificação do número 17,5 usando 14bits (1+5+8), se representado como $0{,}100011_2 \times 2^5$

  0 00101 10001100

  sinal expoente mantissa
- Também se consegue representar números muito maiores do que nos inteiros
  - Exemplo: 65536 = 1 0000 0000 0000 0000_2 (17 bits)

  se representado como $0{,}1_2 \times 2^{17}$

  fica 0 10001 10000000

  sinal expoente mantissa
- Nota: mas como fica 8195?

  10000000000011_2 (14bits)

  $0, 10000000000011_2 \times 2^{14} \rightarrow$ 0 01110 10000000 ← **perdemos precisão**

  o que fica representado é 10000000000000_2!

  (8192)

Normalização da representação

- Não há uma representação única para um número (17,5)
  - 0 00101 10001100 = $0{,}100011 * 2^5$
  - 0 00110 01000110 = $0{,}0100011 * 2^6$
  - 0 00111 00100011 = $0{,}00100011 * 2^7$
  - ...

são todas equivalentes

- Uma convenção é de que o bit a 1 mais significativo da mantissa é sempre o das unidades
  - $17{,}5 = 1{,}00011 * 2^4$
  - Como este bit é sempre 1 até se escusa de representar e ganha-se um bit na mantissa

    exemplo: 0 00100 00011000

Deslocamento do expoente (1)

- Não há expoentes negativos? Como representar
  $0{,}25 = 1_2 \times 2^{-2}$?
- Podíamos usar uma representação em complemento para 2; mas também se pode usar um expoente **deslocado (*biased*)**
  - A ideia é que a representação seja sempre positiva facilitando comparações.
  - Portanto, soma-se sempre uma constante positiva ao expoente, que será próxima de metade do valor máximo representável.
  - No nosso exemplo (5 bits para expoente), usamos 16 (expoente ficará em **deslocamento 16**). Zero é representado por 16 e um valor menor que 16 corresponde a um expoente negativo.
  - Valores do expoente todo a 0 ou todo a 1 são reservados para casos especiais (0 ou infinito)

Deslocamento do expoente (2)

- $17{,}5 = 1{,}00011_2 \times 2^4$ com expoente não deslocado (com bit da mantissa escondido):
  0 00100 00011000
  sinal expoente mantissa
- $1{,}00011_2 \times 2^4$ com expoente deslocado, representa-se o expoente como 4+16 = 20:
  0 10100 00011000
  sinal expoente mantissa
- Para representar $0{,}25 = 1_2 \times 2^{-2}$ representa-se o expoente como -2+16 = 14
  0 01110 00000000
  sinal expoente mantissa

Exemplo ("nossa" representação em 14 bits)

- Exemplo 0,03125 na "nossa" representação de vírgula flutuante:
  $0{,}03125_{10} = 0{,}00001_2 \times 2^0 = 1{,}0_2 \times 2^{-5}$
  - sendo positivo, o bit de sinal é 0;
  - usando o deslocamento 16 o campo para o expoente fica com -5+16=11;
  - escondendo o 1 mais significativo da mantissa fica:
    0 01011 00000000

A norma IEEE-754 Floating Point Standard (1985)

- Precisão simples 32 bits -- float
  - Sinal 1 bit
  - Expoente 8 bits (deslocamento 127)
  - Mantissa 23 bits (c/ bit unidade omitido)
- Precisão dupla 64 bits -- double
  - Sinal 1 bit
  - Expoente 11 bits (deslocamento 1023)
  - Mantissa 52 bits (c/ bit unidade omitido)
- Adoptado quase universalmente

Representações especiais

- Expoente=255 (11111111_2) e mantissa= 0
  - + infinito
  - - infinito
- Expoente=255 e mantissa <> 0
  - NaN (Not a Number)
- Dois valores para 0:
  - Sinal = 0; expoente = 0, mantissa = 0
  - Sinal = 1; expoente = 0, mantissa = 0
- *Semelhante para os 64 bits*

Características dos números representáveis

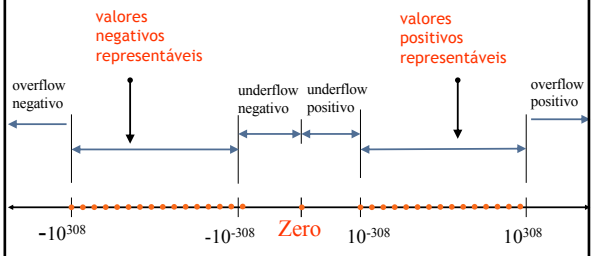
	Precisão IEEE simples	dupla
Bit de sinal	1	1
Bits de expoente	8	11
Bits de mantissa	23	52
Total de bits	32	64
Expoente	excesso 127	excesso 1023
Gama expoentes	-126 a +127	-1022 a +1023
Menor número abs.	2^{-126}	2^{-1022}
Maior número abs.	$\sim 2^{128}$	$\sim 2^{1024}$
em decimal	*$\sim 10^{-38}$ a 10^{38}*	*$\sim 10^{-308}$ a 10^{308}*

Erros de representação de reais

- O conjunto dos reais é contínuo: entre quaisquer dois números há um número infinito de reais
  - com um número limitado de bits de representação só podemos representar um número finito de valores.
- A representação é assim uma aproximação; quanto mais bits tivermos melhor a aproximação
  - Não consegue representar valores demasiado grandes ou valores demasiado pequenos

Valores representáveis

- Número limitado de reais representáveis
  - Exemplo: precisão dupla (64bits)



Exemplo

- Representar 0,1
  - Base 10 → 1/10 = 0,1 fácil!
  - Base 2 → 1/16 + 1/32 +1/256 + ... = $0,000110011(0011)_2$

Erros de representação de reais (2)

- Nas regiões representáveis:
  - Procurar o número de vírgula flutuante mais aproximado do número real que se quer
  - Este arredondamento introduz um erro
- O erro relativo (%):

  $\frac{\text{diferença entre números (erro)}}{\text{número pretendido}}$
- Se aumentar o número de dígitos:
  - de *m*, aumenta a densidade de pontos nas regiões representáveis: maior precisão
  - de *e*, aumenta o tamanho das regiões representáveis: maior gama de valores

Erros de representação de reais (3)

- O nosso formato de 14 bits representa números de $-1,111111111_2 \times 2^{15}$ a -1×2^{-15}, o 0 e de 1×2^{-15} a $1,111111111_2 \times 2^{15}$
- Não consegue representar 2^{-19} ou 2^{128} por exemplo
- Também não consegue representar 128,25 pois é $1000\ 0000,01_2$
  - mas os 9 bits não cabem nos 8 bits da mantissa
  - podemos aproximar com o número 128,0

Erro absoluto é: 0,25 O erro relativo é:

(128,25 – 128,0)/128,25 ≈ 0,2 %

Propagação de erros

- Os erros podem acumular-se; os algoritmos devem tentar minimizá-los
- 1º exemplo (nossa rep. de 14bits):

  128,25*128,25 = 16448,0625

por aproximação de 128,25 a 128:

128*128 = 16384

(entra erro=0,25 ; sai erro=64,0625 → ~0,4%)

Propagação de erros

- Os erros podem acumular-se; os algoritmos devem tentar minimizá-los
- 2º exemplo (nossa rep. de 14bits):

  (512+1)/8 = 64,125 = (512/8)+(1/8)

  513 = 1000000001_2 não é representável
  Logo, neste exemplo, não se deve fazer a soma primeiro;

  512/8 e 1/8 são representáveis
  Por isso devia-se reescrever a expressão nessa forma