# Programación : repaso de C (1)

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS

Agosto 2009

# Outline



# Outline



# Programa

1. Repaso de C (3-4 sesiones).
2. Programación orientada objetos : conceptos (1-2 sesiones).
3. C++ (14 sesiones).
4. Temas selectos...

La idea es adquirir un bagaje suficiente para poder programar eficientemente en cualquier tema de tesis.

# Evaluación

| Tipo | Frecuencia | Porcentaje de la evaluación final |
|---|---|---|
| Tareas | $\approx$ en cada sesión | 40 % |
| Proyecto(s) | en dos partes | 30 % |
| Examenes | dos | 30 % |

# Alrededor de la clase: preguntas y ayudantía

Información sobre la clase centralizada en mi www:
`http://www.cimat.mx/~jbhayet/CLASES/PROGRAMACION/`,

- tareas y clases en PDFs,
- notificaciones diversas, errata sobre las tareas...

# Alrededor de la clase: preguntas y ayudantía

- Ayudante: Hugo Carlos Martínez.
- No dejar pasar las dudas, bajo ningún pretexto. Pre-gun-tar.
- Preguntarme a mí o a Hugo.
- Mail; tél;

# Alrededor de la clase: tareas

- Llegarán en la tarde de la clase $\approx$ 15h.
- Tienen 1 semana para terminarlas (redondeado al día siguiente), por ejemplo:

  `jueves 7 octubre, 15h45 -> jueves 14 octubre, 23h59.`

- Penalidad para retraso: -0.5pt/día.
- Usar herramientas que dejan compatibilidad absoluta (como en el prope) para facilitar la evaluación, evitar el uso de *frameworks* integrados como en Borland...
- Unas soluciones : CodeBlocks,...

# Alrededor de la clase: seminarios

Es importante asistir

- hacer un *break* y abrirse el espíritu,
- adquirir una cultura científica,
- ver conceptos, paradigmas de unas áreas muy diferentes de la tuya que te pueden ayudar en tu investigación,
- pensar en la tesis de maestría para no tener que decidir al último momento...

# Alrededor de la clase: ingles

Es muy importante NO menospreciar la clase de ingles

- es un requisito de la maestría,
- es imposible hacer investigación seria sin un buen nivel en ingles,
- es un plus importante para buscar trabajo,
- Guanajuato siendo muy cosmopolita ¿cómo le van a hablar a esas bonitas chica(o)s daneses o noruega(o)s si no en ingles?

# Alrededor de la clase: otros

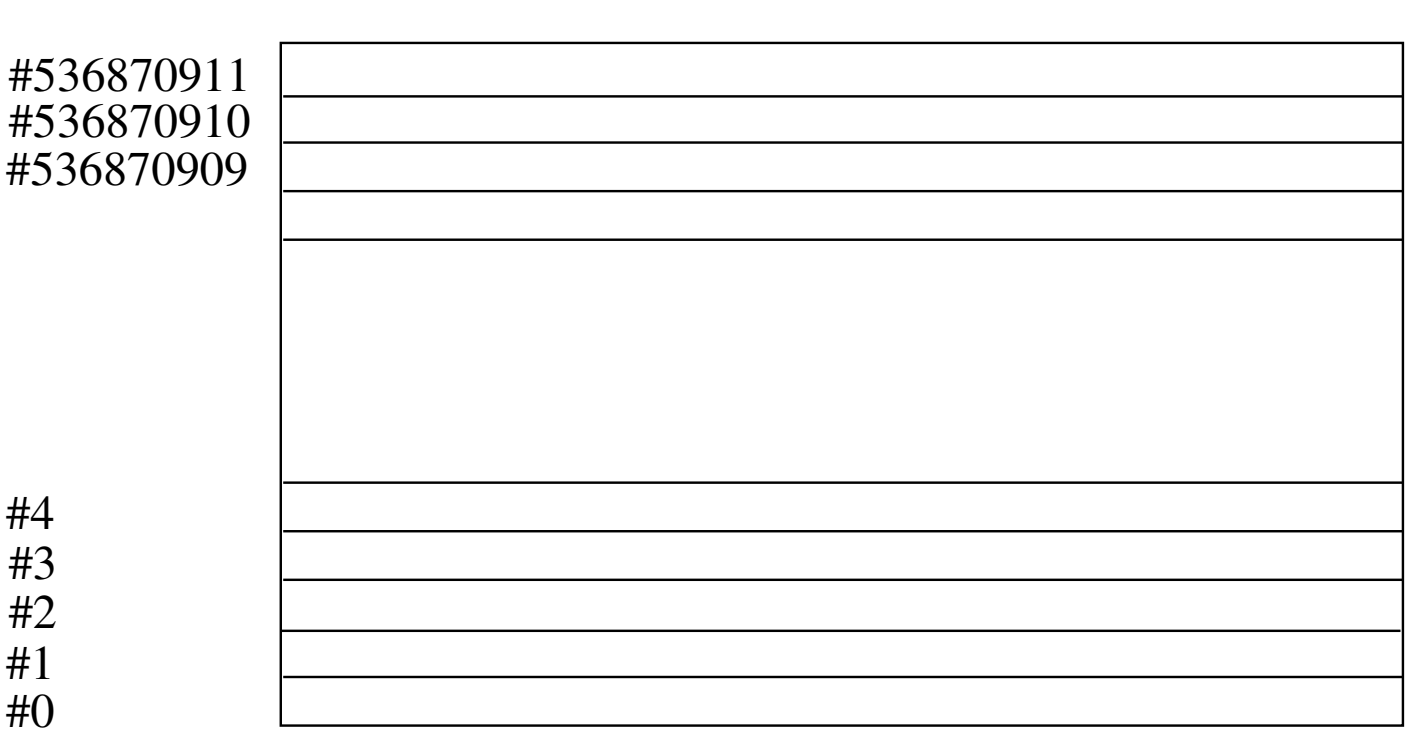
- El semestre va a ser difícil, pero todos los de los semestres pasados lo hicieron, y (en general) lo hicieron bien;
- Organizar bien su tiempo.
- Guardarse tiempo (un jardín secreto) para actividades extra-escolares, de deportes, artes...
- Have fun!

# Outline



# Tipos de datos

Se puede ver la memoria de la computadora como una serie de bytes (octetos, en general), componentes esenciales que se puede direccionar; cada byte tiene una dirección única en la memoria (identificada por un numero en 32 bits, en máquinas 32 bits).



# Tipos de datos

Punto de vocabulario: byte $\neq$ bit,

- *addressable unit of data storage large enough to hold any member of the basic character set of the execution environment*,
- viene de "to bite" y añadieron la y para no confundir con bit,
- en general, 1 byte = 1 octeto = 8 bits.

# Tipos de datos

Para verificar, buscar el archivo LIMITS.H en tu sistema:

```
Bidarray[~][15:47]>more /usr/include/i386/limits.h

#ifndef _I386_LIMITS_H_
#define _I386_LIMITS_H_

#include <sys/cdefs.h>
#include <i386/_limits.h>

#define CHAR_BIT     8     /* number of bits in a char */
```

# Tipos de datos

- Las máquinas pueden procesar varios bytes al mismo tiempo (según la capacidad de sus registros) : los paquetes de bytes que pueden procesar son los *words*.
- Hoy, en la mayoría de los casos, son de 32 bits pero hay mas y mas maquinas de 64 bits.
- En general, los sistemas k-bits tienen registros y *buses* de k bits (*hardware*), y sistemas de explotación que manipulan direcciones en memoria de k bits (*software*). Se puede tener una maquina 64-bits y un OS 32-bits. Pero no al revés.

# Tipos de datos

- El tipo define dos cosas : el numero de octetos que se va a usar para el dato y la manera de usar cada uno de los octetos.
- Los tipos elementales son los caracteres, enteros y flotantes (números reales).
- No hay estándar en cuanto al tamaño de los tipos.

# Datos

En The C++ Programming Language, 2nd Edition

```
This is what is guaranteed about sizes of fundamental types:

1 == sizeof(char) <= sizeof(short) <= sizeof(int)
                                   <= sizeof(long)
sizeof(float) <= sizeof(double) <= sizeof(long double)
sizeof(I) == sizeof(signed I) == sizeof(unsigned I)

where I can be char, short, int or long. In addition, it is
guaranteed that a char has at least 8 bits, a short at least
16 bits and a long at least 32 bits... assuming more is
hazardous.
```

# En una maquina 32 bits típica

| Tipo | Tamaño (en *bytes*) | Valores |
|---|---|---|
| char | 1 | $[-128, 127]$ |
| short | 2 | $[-32768, 32767]$ |
| int | 4 | $[-2147483648, 2147483647]$ |
| long | 4 | $[-2147483648, 2147483647]$ |
| float | 4 | $[1.18 \times 10^{-38}, 3.4 \times 10^{38}]$ |
| double | 8 | $[2.2 \times 10^{-308}, 1.8 \times 10^{308}]$ |
| long double | 10 | $[1.18 \times 10^{-4932}, 3.4 \times 10^{4932}]$ |
| apuntadores | 4 | $[0, 2^{32} - 1]$ |

- *unsigned* : solo valores positivos.

# En una maquina 64 bits

Todavía no hay estándar único!

| Tipo | Tamaño (en *bytes*) |
|---|---|
| char | 1 |
| short | 2 |
| int | 4 |
| long | 4/8 |
| apuntadores | 8 |

# sizeof

Da el numero de bytes necesarios para el tipo/la variable pasado en argumento,

- sizeof(int),
- sizeof($x$), donde $x$ es una variable.

# Tipos enteros

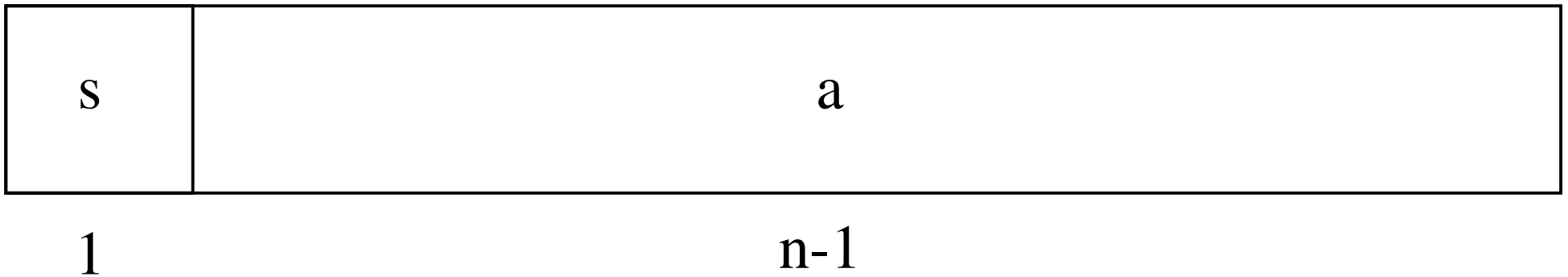
- Para representar un subconjunto de $\mathbb{N}$ ("unsigned").
- Representación de los positivos en la base binaria, sobre $n$ bits

$$a = \sum_{i=0}^{n-1} a_i 2^i.$$

- ¿Rango?

# Tipos enteros

Para representar un subconjunto de $\mathbb{Z}$:




Si tenemos un tipo entero con $n$ bits (8, 16, 32...),

- Marcar el signo con el bit de peso fuerte : $s = 0$ para positivos (así se puede identificar fácilmente si un entero es positivo o negativo).
- Los positivos representados en la base binaria sobre $n - 1$ bits,

$$a = \sum_{i=0}^{n-2} a_i 2^i.$$

- ¿Como representar los negativos ?

# Tipos enteros

¿Representar negativos ?

- Nos gustaría pasar rápido de un entero a su negativo.
- Nos gustaría que $-(-a) = a$, que $a + (-a) = 0$.
- Notar que una adición es mucho mas fácil que hacer que una sustracción.
- Nos gustaría tener operaciones aritméticas rápidas; idealmente, una sustracción ES una adición, no queremos tener casos:

$$a + (-b) = a - b$$

- Representación signo+valor absoluta. ¿Problemas ?

# Tipos enteros

Representar negativos ? Idea es usar el complemento!

- Representación complemento a uno. ¿Problemas ?

$$a = \sum_{i=0}^{n-1}(1 - a_i)2^i = 2^n - 1 - |a|.$$

- Representación complemento a uno mas uno (complemento a dos),

$$a = \sum_{i=0}^{n-1}(1 - a_i)2^i + 1 = 2^n - |a|.$$

# Tipos enteros

- No distinción entre substracción y adición.
- Representación única de 0.
- ¿Rango función de $n$ ?
- ¿Cómo se representan 131 y -131 en un short ?

# Tipos enteros

Complemento a dos:

- calculo rápido: después del primer uno (a excluir), tomar el complemento a uno de todos los dígitos,
- en la adición, detección de *overflow*, ¿una manera rápida de detectarla ?

# Representación de reales


- Punto fijo : se usa cuando uno quiere trabajar con una precisión absoluta dada.
- Representación :

$$z = sa + d2^{-p}.$$

- ¿Rango?
- No hay soporte de reales a punto fijo por default en C (librerías especializadas).

# Reales en punto fijo

Ejemplo : Representar 78.345 (decimal) con $n = 16$ y $p = 6$.

# Reales en punto fijo : mas y menos

+ Permiten precisión arbitraria, la que quieres (o que quiere tu cliente/jefe).
+ Rápido (fundamentalmente similar a enteros).
- No muy flexible.
- Se gasta espacio para pequeños números o números con pocas cifras después del punto.
- "Consume" muchos bits : para resolución de $\frac{1}{65536} \approx 1.5\ 10^{-5}$ se necesitan 16 bits después del punto !

# Reales flotantes


- Norma IEEE 754.
- Representación :

| Elemento | Signo | Exponente | Mantisa |
|---|---|---|---|
| float | 1 | 8 | 23 |
| double | 1 | 11 | 52 |

$$f = s.m.b^e$$

$$e = k - (2^{q-1} - 1)\ ;\ 0 \leq m < b$$

- Ambigüedad ?

# Reales flotantes

- Mantisa normalizada : un sólo dígito no nulo antes de la coma (=1 si $b = 2$)

$$1.0101001$$

$$8.8178582$$

- En binario, tenemos efectivamente 24/53 bits para representarla con $1 \leq m < 2$ y $b = 2$, el carácter 1 esta implícito

$$f = s.(1 + m.2^{-(n-1-q)}).2^e$$

- Rango del exponente :

$$-2^{q-1} + 1 \leq e \leq 2^{q-1}$$

# Reales flotantes

- Ejemplo : representar todos los reales flotantes positivos con $b = 2$, $q = 2$, $n = 5$.
- ¿Representar 0?

# Flotantes : casos especiales

- La norma $\mathrm{IEEE}\ 754$ define tres números adicionales : *NaN*, $+\infty$, $-\infty$ que permite manejar operaciones de tipo $1/0\ldots$
- ¿Como representarlos, ademas del 0 ?
- Solución : usar los dos valores extremos posibles del exponente para representarlas. Caso de los *float* :

| $e = k - 127$ | $m$ | $f$ |
|---|---|---|
| $-126 \leq e \leq 127$ | $0 \leq m_f < 2^{23}$ | $\pm 1.m \times 2^e$ |
| 128 | 0 | $\pm\infty$ |
| 128 | $\neq 0$ | *NaN* |
| $-127$ | 0 | $\pm 0$ |
| $-127$ | $\neq 0$ | $\pm 0.m \times 2^{-126}$ |

# Flotantes : casos especiales

- Caso de los *double* :

| $e = k - 1023$ | $m$ | $f$ |
|---|---|---|
| $-1022 \leq e \leq 1023$ | $0 \leq m_f < 2^{52}$ | $\pm 1.m \times 2^e$ |
| 1024 | 0 | $\pm\infty$ |
| 1024 | $\neq 0$ | *NaN* |
| $-1023$ | 0 | $\pm 0$ |
| $-1023$ | $\neq 0$ | $\pm 0.m \times 2^{-1022}$ |

# Flotantes : casos especiales

- Remarcar el valor binario de número que representa 0.
- Los casos de $\pm\infty$ y *NaN* corresponden a exponente a 11111....
- Con exponente nulo y mantisa no nula, son números en *underflow*.

# Flotantes : límites

- Overflows, underflows.
- Redondeos.

# Flotantes : el crash de Ariane 5


# Flotantes : el crash de Ariane 5

- El primer Ariane 5 exploto en 1996 (500 millones de dólares evaporados) porque los sensores inerciales dejaron de enviar datos : estaban reseteando a causa de un fallo a causa de una excepción no procesada.
- Esta excepción fue generada por un *overflow* : una velocidad estaba guardada como double (64 bits), y estaba convertida a un short en un pedazo de programa. Los ensayos habían sido hechos con datos de Ariane 4 (que era menos rápida). Hubo una excepción pero no había mecanismo para tratarla.
- Mas informacion aqui

# Flotantes : redondeo

Todos los reales no pueden representarse correctamente : por ejemplo, 0.1 (decimal) necesita una infinidad de dígitos en un sistema binario !

# Flotantes : redondeo

Varias maneras de redondear un real $x$ tal que $x^- \leq x \leq x^+$, con $x^-$ y $x^+$ reales flotantes en una representación dada

- Hacia $-\infty$.
- Hacia $+\infty$.
- Hacia 0.
- Hacia el mas cercano (en caso de igualdad, tomar el flotante de mantisa par).

Una vez elegida una manera de redondear, el resultado de una operación dada es especificado (redondeo correcto). La norma impone el redondeo correcto para la adición, la substracción, la multiplicación, la división y la raíz cuadrada.

# Flotantes : redondeo

- Con redondeo correcto, llegamos a poder dar un intervalo para el resultado exacto de cada operación.
- Portabilidad.
- Operaciones como log o cos no incluidas en la norma.

# Flotantes : redondeo

En 1991, un misil Patriot fue lanzado para interceptar un Scud, durante la Primera Guerra del Golfo. El cálculo de la fecha dentro del sistema de navegación del misil estaba hecho por mútiples de $\frac{1}{10}$ de segundos. Llevaba por consecuente una aproximación redondeada de un décimo de segundo, sobre 24 bits : $209715.2^{-21}$, error de $10^{-7}$ segundos cada décimo de segundos. El misil había sido prendido como 100h antes de llegar a su meta... Error acumulada $0.34s$ (demasiado). Balance : 28 muertos.

Con una mejor aproximación sobre 24bits (que sí es posible) hubiera podido alcanzarle...

# Flotantes : doble redondeo

Considera $x = 1.0110100000001$ .

- Redondeado mas cercano de x a 9 bits : $y = 1.01101000$
- Redondeado mas cercano de y a 5 bits : $z = 1.0110$
- Redondeado mas cercano de x a 5 bits : $z' = 1.0111$ !!

Es un problema en Linux con los FPUs de los procesadores $x86$ que trabajan internamente con doble precisión extendida (64 bits de mantisa) que es luego convertida a doble precisión.
Problema solo con redondeo al mas cercano.

# Flotantes : representabilidad

- En *double*, el flotante mas pequeño representable es $x = 1.0 \times 2^{-1022}$.
- Considera $y = (1 + 2^{-52}) \times 2^{-1022}$, ¿es representable ?
- Ahora considera $x - y$, ¿a que valor esta redondeado ?
- Que pasaría con :

```
if (x!=y)
        z = 1.0/(x-y);
```

- Hacer los tests sobre $x - y$ !

```
if (fabs(x-y)>epsilon)
        z = 1.0/(x-y);
```

- Por eso los números denormalizados.

# Flotantes

```
#include <stdio.h>
#include <math.h>
int main() {
  float f =  pow(2.0,-23);
  float a = (1.0)*pow(2,-125);
  float b = (1.0+f)*pow(2,-125),z;
  if (a!=b)
    fprintf(stderr,"a and b have different representations \n");
  else
    fprintf(stderr,"a and b have same representations \n");
  fprintf(stderr,"Vals: %e %e %e\n",a,b,f);
  float d = a-b;
  z = 1.0/d;
  unsigned int id   = *(unsigned int *)&d;
  fprintf(stderr,"Diff : %e \n",d);
  fprintf(stderr,"Diff : %lx \n",id);

  unsigned int iz   = *(unsigned int *)&z;
  fprintf(stderr,"Diff inv : %e \n",z);
  fprintf(stderr,"Diff inv : %lx \n",iz);
  return 0;
}
```

# Flotantes

```
Macaye[CLASE1][15:38]>./test2
a and b have different representations
Vals: 2.350989e-38 2.350989e-38 1.192093e-07
Diff : -2.802597e-45
Diff : 80000002
Diff inv : -inf
Diff inv : ff800000
```

Interpretación ?

# Flotantes : error absoluto al redondeo

Es el error sobre el ultimo dígito de la mantisa. Notando $r = n - 1 - q$ (en el caso general), en el caso del redondeo al mas cercano

$$(b/2) \times b^{-r} \times b^{e}.$$

# Flotantes : ulp

Es el error que se hace medido en el ultimo dígito (*unit in the last place*). Por ejemplo, si $z$ esta representado por $d.dd\ldots dd \times b^e$, el error en ulps es

$$|d.dd\ldots dd - (z/b^e)|b^r$$

- Si $r = 3$, $b = 10$, cual es el error en ulps por aproximar .0314159 por $3.14 \times 10^{-2}$ ? por $3.142 \times 10^{-2}$ ?
- Un redondeo al mas cercano tiene como error .5 ulp.

# Flotantes : error relativa

$$(\textit{Valor exacto} - \textit{Valor representado})/(\textit{Valor exacto})$$

Pregunta : ¿a que error relativo corresponde un redondeo (.5 ulp) ?
Los valores están entre $b^e$ y $b \times b^e$, entonces el valor relativo es :

$$\frac{1}{2} b^{-r} \leq \frac{1}{2} ulp \leq \frac{b}{2} b^{-r}.$$

Un error de redondeo estará siempre mayorado por $\frac{b}{2} b^{-r}$ (precisión maquina $\epsilon$).

# Flotantes : calcular diferencias

Cómo se podría diferenciar, en particular con magnitud muy diferentes, como $3.215 \times 10^{12} - 1.25 \times 10^{-5}$ :

- Usar mas dígitos y luego redondear :

$$\begin{aligned} x &= 2.15 \times 10^{12} \\ y &= 0.00000000000000125 \times 10^{12} \\ x - y &= 2.14999999999999875 \times 10^{12} \end{aligned}$$

- Usar numero de digitos fijos (y pues redondear ya desde el principio)

$$\begin{aligned} x &= 2.15 \times 10^{12} \\ y &= 0.00 \times 10^{12} \\ x - y &= 2.15 \times 10^{12} \end{aligned}$$

Shift del operando mas pequeño.

# Flotantes : operaciones aritmeticas

Las adiciones/substraciones se hacen :

- Alineando todo sobre el numero de exponente mayor (shift de los bits).
- Aplicando operación aritmética sobre la mantisa.
- Eventualmente re-alineando todo para normalizar.

# Flotantes : calcular diferencias

Otro caso, $10.1 - 9.93$

$$\begin{aligned} x &= 1.01 \times 10^1 \\ y &= 0.99 \times 10^1 \\ x - y &= 0.02 \times 10^1 \end{aligned}$$

Error relativo : 30 ulps ! (y cada dígito es incorrecto !)

# Flotantes : calcular diferencias

El problema es el de la cancelación : la diferencia puede eliminar cantidad de digits confiables (los de pesos mas altos), dejando los que son los menos confiables, porque han sido contaminados por operaciones de redondeo ! Efecto amplificador de los errores !

# Flotantes : calcular diferencias

Error relativo en el peor de los casos :

$$(b^{-r+1} - b^{-r})/b^{-r} = b - 1$$

Entonces el error absoluto puede ser de orden de magnitud tan elevado como el resultado !!! Puede ser útil añadir un dígito aditivo (guard digit) para evitar esas situaciones (en este caso se muestra que el error relativo no puede ser mayor que $2\epsilon$).

# Flotantes : calcular diferencias

Considerar las formulas de las raíces de polinomios de segundo grado

$$\frac{-b+\sqrt{b^2-4ac}}{2}, \frac{-b-\sqrt{b^2-4ac}}{2}$$

Si $a$, $b$, $c$ son redondeados y que ademas $b^2 >> ac$, tenemos problemas !
Usar re-escritura de las formulas :

$$\frac{2c}{-b-\sqrt{b^2-4ac}}, \frac{2c}{-b+\sqrt{b^2-4ac}}$$

Por cada caso ($b > 0$ o $b < 0$) hay una fórmula mas estable que la otra !

# Flotantes : calcular diferencias

Qué es mejor : calcular $x^2 - y^2$ o $(x - y)(x + y)$ ? Por qué ?

# Flotantes : en regla general

Si sabemos que va a haber problemas de cancelación, analizar el método/algoritmo que usamos para eventualmente evitarlo (factorización...). El orden de las operaciones importa ! Perdemos la asociatividad de las operaciones (adición/substracción), la distributividad de la multiplicación sobre la adición.

# Flotantes : matemáticas raras

Por consecuencia de la absorción de los pequeños números, un teorema de cálculo flotante es que para toda serie positiva tal que $\lim_{n\to\infty} u_n \to 0$, $\sum_n u_n$ es convergente !

# Referencias


- V. Lefèvre y P. Zimmermann, Arithmétique flottante, Rapport de recherches INRIA N.5105
- The GNU C Library, online manual
- David Goldberg. What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys, 23(1):5–48, March 1991.