Representación de números reales y caracteres

Área de Arquitectura y Tecnología de Computadores – Versión 1.0.866, 19/01/2022

Índice

Objetivos

Esta sesión analizará la forma en la que se representan números reales y caracteres en un lenguaje de alto nivel, en concreto en C++. Además, se realizarán diversas operaciones con datos numéricos para entender las repercusiones que tienen los errores de redondeo y desbordamiento en un programa.

Conocimientos y materiales necesarios

Para aprovechar adecuadamente esta sesión de prácticas, el alumno necesita:

- Conocer los sistemas de codificación numéricos de números reales (IEEE-754).
- Conocer los sistemas de codificación de caracteres.
- Conocer el lenguaje de programación C++.

Durante la sesión se plantearán una serie de preguntas que puedes responder en el correspondiente cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916) en el Campus Virtual. Puedes abrir el cuestionario en otra pestaña del navegador pinchando en el enlace mientras mantienes pulsada la tecla `Ctrl`.

1. Codificación de números reales

Vamos a ver cómo se codifican números reales. En primer lugar, codifica el número -27.625 en el formato IEEE 754. ¿Cuál es la codificación en hexadecimal? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 1.

Vamos a comprobarlo con un programa. Sigue estos pasos:

- Descarga el fichero `2-Reales` del Campus Virtual y descomprímelo. Abre el fichero de solución en Visual Studio.
- Modifica el programa insertando al principio del `main` la declaración de una variable llamada `f` de tipo `float`.
- Añade a continuación una sentencia que asigne a la variable `f` el valor `-27.625`.
- Añade un mensaje que imprima por pantalla el valor de la variable.
- Ejecuta el programa y verifica que funciona correctamente.
- Inserta un punto de ruptura en la sentencia que muestra el mensaje e inicia la depuración.
- Muestra la ventana de memoria e introduce en el campo dirección `&f`. En los cuatro primeros bytes deberás ver la codificación de f. Teniendo en cuenta que está guardada usando la convención *Little Endian*, comprueba que coincide con la codificación que habías hecho tú.

2. Redondeo

Los números reales se ven afectados por los mismos problemas que los números enteros en cuanto al desbordamiento, aunque la forma de detectarlo es diferente. Además, en la representación de números reales aparece un nuevo problema: el problema del redondeo. Este problema ocurre cuando un número real no tiene una representación exacta.

Para comprobar en qué consiste el problema de redondeo convierte el número 0.1 a binario. ¿Cuál es su codificación? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 2. El número en binario no tiene una representación exacta con lo que se tendrá que trabajar con aproximaciones.

A continuación se van a comprobar los problemas que pueden generar este tipo de errores:

- En primer lugar comenta todo el código del `main`.
- Añade dos variables de tipo `float` `f1` y `f2`.
- Asigna el valor 0.1 a `f1` y 0.3 a `f2`.
- Imprime por pantalla el valor de ambas variables.
- Compila y ejecuta el programa pulsando `Ctrl` + `F5`.
- Aparentemente se imprime el valor exacto, pero eso se debe a que por defecto se imprimen pocos decimales. Para imprimir los números reales con más decimales en primer lugar hay que añadir al programa un nuevo fichero de cabecera:

```cpp
#include <iomanip>
```

- A continuación se puede indicar a `cout` que se quieren imprimir más decimales de la siguiente forma:

```cpp
cout << "f1: " << setprecision(15) << f1 << endl;
cout << "f2: " << setprecision(15) << f2 << endl;
```

- Ejecuta de nuevo el programa y observa los resultados.

Ciertamente los errores de redondeo son muy pequeños. Sin embargo, si no se tienen en cuenta se pueden generar graves problemas.

- Añade al programa una sentencia condicional que imprima si `f1·3` es igual a `f2` o si son distintos. Dado que estamos trabajando con números reales, todos los números de la expresión deben ser reales para que el compilador no aplique un tipado diferente: la expresión a utilizar debería ser `f1*3.0`.
- Ejecuta el programa y comprueba los resultados.

El problema es que los errores de redondeo cometidos al representar `f1` y hacer la operación `f1*3.0` son distintos a los errores cometidos al representar `f2`, lo que hace que las magnitudes comparadas sean ligeramente diferentes. Por este motivo para comparar dos números reales se utiliza siempre una expresión similar a la siguiente:

CPP

```
const float TOLERANCIA = 0.0000001;
if (fabs(f1*3.0 - f2) < TOLERANCIA)
  cout << "Son iguales" << endl;
else
  cout << "Son distintos" << endl;
```

- Añade el fragmento de código al programa y verifica que ahora la comparación da el resultado esperado. La función `fabs` calcula el valor absoluto de un número y está definida en el archivo de cabecera `math.h`. El valor de la tolerancia depende del tipo de datos que se manejen en la aplicación.

Un error de redondeo muy pequeño puede dar lugar a un error muy grande si se realizan muchas operaciones con números reales. Los pequeños errores se van acumulando y finalmente el error empieza a ser significativo.

Supongamos que se almacena sobre una variable de tipo `float` el número de euros que una persona tiene en el banco. ¿Qué ocurriría si dicha persona realiza 20 millones de ingresos de 10 céntimos cada uno?

- Añade al programa el siguiente fragmento de código que simularía el caso anterior:

```cpp
float euros = 0;
const int UN_MILLON = 1000000;
for (int i = 0; i < 20*UN_MILLON; i++)
  euros = euros + 0.1;
cout <<"euros = " << euros << endl;
```

- 20 millones multiplicado por 0.1 debería dar como resultado 2000000 (2 millones). Ejecuta el programa y observa el error que se produce.
- Piensa que ocurriría si en lugar de ingresar 0.1 euros se ingresase 1 euro cada vez. En este caso el número 1 tiene una representación exacta donde no se producen errores de redondeo. ¿Crees que el resultado será correcto?
- Haz los cambios oportunos y comprueba lo que ocurre en este caso.
- Nuevamente aparecen errores de redondeo. Aunque el número 1 tiene una representación exacta, no ocurre lo mismo con otros números que aparecen en los cálculos.
- Añade el siguiente código al programa:

```cpp
float f3 = 19*UN_MILLON;
cout << "f3: " << setprecision(15) << f3 << endl;
f3 = f3+1;
cout << "f3 + 1: " << setprecision(15) << f3 << endl;
```

- Reflexiona sobre el resultado que va a producir el código que se acaba de añadir. Ejecuta el programa y analiza los resultados.

Por todos estos motivos, la utilización de números reales en los programas sin conocer sus limitaciones debidas a los errores de redondeo puede provocar grandes problemas. Es por tanto esencial para el desarrollo de programas robustos el conocimiento de la forma en la que se codifica la información.

3. Codificación de caracteres

Vamos a ver a continuación cómo se codifican caracteres en C++. Sigue estos pasos:

- Comenta el código de la función `main`.
- Define una variable `c1` de tipo `char` y asígnale el valor `'a'`. Fíjate que para indicar un carácter individual se utilizan las comillas simples.
- Define otra variable `c2` también de tipo `char` y haz que contenga el carácter `'ñ'`.
- Añade una sentencia que imprima los dos caracteres separados por un espacio.
- Ejecuta el programa y comprueba que la `a` se imprime correctamente pero la `ñ` no.

Vamos a analizar lo que está ocurriendo. En primer lugar, debes saber que el tipo `char` de C sirve para almacenar un byte. Vamos a ver en concreto qué secuencia de bits se almacena en las dos variables que hemos definido:

- Pon un punto de ruptura en la línea que imprime las variables e inicia la depuración con F5.
- Muestra la ventana de memoria y haz que muestre el valor de la variable `c1` en hexadecimal. ¿Cuál es? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 3. Como puedes comprobar si buscas en Internet la tabla de códigos ASCII (http://es.wikipedia.org/wiki/ASCII) ese valor hexadecimal se corresponde con la codificación ASCII de la letra `a`. Como es una letra del alfabeto inglés y forma parte del ASCII, también tendrá esa misma codificación en todas las extensiones del ASCII, incluyendo ISO-Latin-1, y en UTF-8.
- Muestra en la ventana de memoria el valor de la variable `c2` en hexadecimal. ¿Cuál es? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 4. Este no puede ser un código ASCII porque los códigos ASCII sólo van hasta 127 (7Fh). En realidad es un código que se corresponde con una de las extensiones de ASCII. MS-DOS, el sistema operativo de Microsoft antecesor de Windows, tradicionalmente usaba en España una extensión llamada página de códigos 850 (http://es.wikipedia.org/wiki/Página_de_códigos_850). Sin embargo, en la actualidad las aplicaciones usan distintas extensiones según estén configuradas, siendo una de las más comunes en España la denominada Windows-1252 (http://es.wikipedia.org/wiki/Windows-1252). ¿Cuál de estas dos extensiones asigna a la `ñ` el código que se encuentra almacenado en memoria? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 5.

Esa es la versión que está utilizando el editor de Visual Studio. Sin embargo, la consola que se muestra cuando ejecutas el programa está utilizando la otra extensión y, por esta razón, el carácter mostrado no se corresponde con la `ñ`, sino con el carácter que se corresponde con el número `F1h` en esa extensión.

Vamos a comprobar que es así:

- Finaliza la depuración.
- Modifica la sentencia de asignación de `c2` para que se le asigne el código hexadecimal correspondiente a la `ñ` en la extensión que utiliza la consola. ¿Cuál es? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 6. Para asignar directamente un código hexadecimal puedes poner su valor en hexadecimal antecediéndolo de 0x. Por ejemplo, si quieres asignar el valor hexadecimal `5a` a la variable `var` tendrías que poner `var = 0x5a`.
- Ejecuta el programa y comprueba que ahora se imprime la `ñ` correctamente.

4. Ejercicios adicionales

✓ Es importante darse cuenta de que los errores con números reales que se han observado en esta práctica ocurrirán en la mayoría de los programas que trabajen con datos numéricos. A modo de ejemplo, abre una hoja Excel y añade la siguiente información a sus celdas:

- En la celda `A1` escribe 1,324.
- En la celda `A2` escribe 1,319.
- En la celda `A3` calcula la diferencia entre `A1` y `A2` de la siguiente forma: =A1-A2.
- En la celda `A4` comprueba que la diferencia es 0.005 de la siguiente forma: =SI(A3=0,005;"Iguales";"Diferentes").
- Interpreta el resultado. Deberías ser capaz de identificar el problema.

¿Se podría utilizar una variable de tipo `char` para almacenar cualquier carácter codificado con UTF-8? ¿Por qué? Responde en el cuestionario (https://www.campusvirtual.uniovi.es/mod/quiz/view.php?id=143916): pregunta 7.