Actas de las XX JENUI. Oviedo, 9-11 de julio 2014
ISBN: 978-84-697-0774-6
Páginas: 213-217

C-Sim, un simulador de manejo de memoria de C/C++

Baltasar García Perez-Schofield
Departamento de Informática
Universidad de Vigo
Ourense
jbgarcia@uvigo.es

Francisco Ortín Soler
Departamento de Informática
Universidad de Oviedo
Oviedo
ortin@uniovi.es

Resumen

La materia de Programación es parte de la formación obligatoria para alcanzar el grado en Ingeniería Informática. En la Escuela Superior de Ingeniería Informática de la Universidad de Vigo, se ha apostado por una enseñanza más tradicional (*imperative-first*), de la programación: frente a la estrategia de empezar directamente con programación orientada a objetos, llamada *objects first.* Inicialmente se realiza una primera parte (la asignatura Programación I), en C++ (sin utilizar clases y objetos), y a continuación, en Programación II, se explora el paradigma orientado a objetos con el mismo lenguaje. Esto hace que los estudiantes se enfrenten a conceptos, como el de manejo de la memoria, a un nivel de abstracción más cercano a la máquina. Así, deben comprender conceptos como punteros, bloques de memoria, liberación de memoria... etc. C-Sim nace como una herramienta de ayuda a la docencia, con la que el profesor puede poner fácilmente ejemplos en clase, y con la que los estudiantes pueden experimentar en horas no presenciales.

Abstract

The subject of Programming is part of the mandatory formation for undergraduate students of Computer Science Engineering. At the Escuela Superior de Ingeniería Informática of the University of Vigo, we have taken the path of dividing this formation in two actual subjects: Programación I, for procedural programming, and Programación II, focused in object-oriented programming. In contrast to the strategy called objects first, we still use a imperative first approach. Both subjects use the same vehicular programming language: C++. This forces students to understand programming concepts, such as memory management, from a quite low abstraction level: pointers, memory blocks, memory release, etc. C-Sim was created as an assistance tool for the lecturer to be able to create examples in the classroom, while the students can experiment with the tool in their own.

Palabras clave

C, C++, Punteros, Lenguajes, Programación.

1. Motivación

Los estudiantes deben enfrentarse a multitud de retos a la hora de comprender y estudiar los conceptos impartidos en los primeros años de carrera. En concreto, en los grados en Ingeniería Informática, en el primer curso se concentran un buen porcentaje de la materia de Programación, es decir, buena parte de las asignaturas que tienen directamente que ver con la programación y los lenguajes de programación.

En el grado de Ingeniería Informática de la Universidad de Vigo, impartido en la Escuela Superior de Ingeniería Informática, se tomó la vía de mantener una enseñanza más tradicional, basada en la comprensión de la programación estructurada, y posteriormente, el abordaje de la programación orientada a objetos.

Los alumnos se enfrentan entonces a una visión de la programación y todos los conceptos relacionados (como el manejo de la memoria) desde el bajo nivel de abstracción que ofrecen C y C++. Si bien este último lenguaje de programación ofrece referencias, estas por si mismas no cubren todos los posibles escenarios en los que se precisa crear y manejar colecciones de objetos. Entonces se hace necesaria la impartición del concepto de puntero, y con ellos, muchos conceptos de bajo nivel asociados al manejo de la memoria.

Los profesores hemos detectado varios conceptos complejos, de difícil asimilación por parte de los estudiantes. En primer lugar, el concepto de memoria como un vector unidimensional, y la codificación de distintos tipos, con diferentes tamaños, en dicha memoria. También el concepto de puntero como una variable que guarda una dirección de memoria, y el número de bytes a los que puede afectar (basado en el tipo de dicho puntero); el acceso a la zona de memoria afectada por un puntero, y finalmente el uso de aritmética de punteros (principalmente para el recorrido de vectores y matrices).

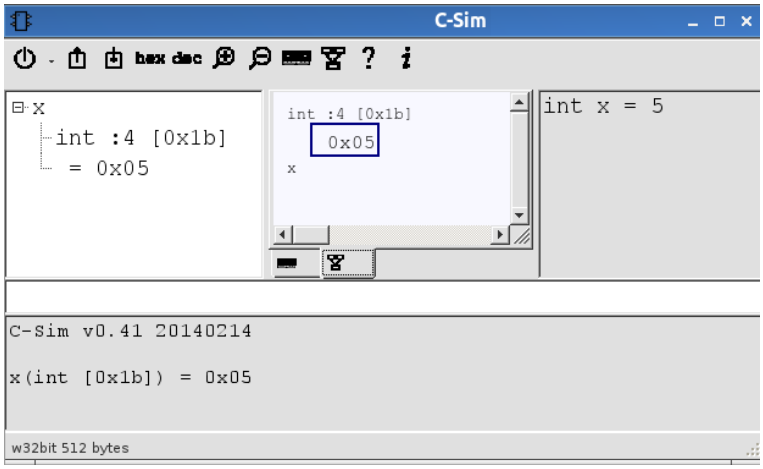

Figura 1: El entorno de trabajo presentado por C-Sim.

Es en este escenario en el que surge el planteamiento de crear una herramienta que permita de forma sencilla simular el comportamiento en memoria de un puntero, de manera que se pueda interactuar y observar el efecto del empleo de punteros, de realizar cambios en variables o en los mismos punteros, y finalmente, del efecto de los operadores & y *, de la forma más educativa posible.

El resto de este documento se estructura como sigue: primero se realiza una discusión exhaustiva de las posibilidades de la herramienta, para después analizar su implementación, así como el análisis de otras propuestas existentes; y finalmente se presentan las conclusiones y trabajo futuro.

2. C-Sim como ayuda a la docencia

En esta sección se discuten en profundidad las funcionalidades y el diseño de la interfaz de usuario que las soportan.

2.1. El entorno de trabajo

C-Sim[1] presenta un entorno de trabajo ordenado (mostrado en la Figura 1), en el que a la izquierda se mantiene un árbol con diferente infomación relacionada con las variables creadas (por cada variable se muestra su tipo (int), su tamaño en bytes (4) y su dirección de memoria (0x1b); adicionalmente se muestra su valor (0x05)). En el centro, se puede cambiar entre un panel con el diagrama que representa la situación en memoria actual, y la representación de la memoria en celdas. A la derecha, un histórico de las instrucciones introducidas ayuda a recordar los pasos necesarios para llegar a la situación actual. Finalmente, la parte inferior presenta un cuadro donde se visualiza el efecto directo en memoria de cada instrucción que el usuario introduce.

2.2. Manejo y características principales

La herramienta soporta un subconjunto reducido del lenguaje de programación C [2], así como toma también las referencias de C++ [3]. Se permiten los tipos **int** y **char**, los punteros a dichos tipos, y también las referencias; el acceso a un puntero, y la obtención de la dirección de cualquier variable. Se pueden asignar literales numéricos y de carácter. C-Sim permite guardar estos pequeños programas como archivos de texto, que se pueden recuperar más tarde. Estos archivos de texto se podrían convertir fácilmente en programas C/C++.

Un ejemplo podría ser el siguiente: se crea una variable de tipo entero llamada *x* con valor 5, y se la hace apuntar por un puntero llamado *ptrX*.

```
int x = 5
int * ptrX = &x
```

Las instrucciones se pueden introducir con o sin punto y como final: si bien esto es obligatorio en C++, en esta herramienta cada sentencia se debe introducir completa en cada entrada, por lo que no es necesario.

Con solamente estas dos sentencias, el usuario puede observar un diagrama aclarativo en el que cada variable ocupa una caja, en la parte superior de esa caja se indica el tipo, tamaño en *bytes*, y posición de dicha variable en la memoria, y en la parte inferior su nombre. Existirán dos de estas cajas, como se observa en la Figura 2, y estarán unidas por una flecha. Cuando esto sucede, se puede observar que el valor del puntero coincide con la posición en memoria de la variable apuntada. Al cambiar a la vista de la memoria, pulsando en el árbol de variables, se puede acceder directamente a la posición de memoria ocupada por una variable con un simple click de ratón.

[1] C-Sim puede ser descargado libremente de http://webs.uvigo.es/jbgarcia/prys/

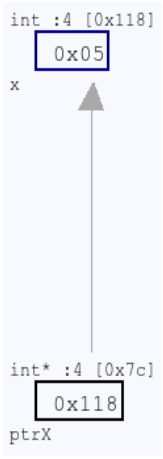

Figura 2: Diagramador mostrando una variable señalada por un puntero.

Es interesante observar aquí que la posición en la que se encuentra *x* contiene un cero en su primer, segundo y tercer byte, así como un cinco en su *byte* final. Se asume que el tamaño de una variable **char**, como de hecho señala el estándar de C++ [3], siempre ocupa un solo *byte*. El tamaño de una variable entera se asume que siempre es igual al ancho de palabra del ordenador (32*bit*, por defecto). El estándar de C++ solo señala que un entero ocupa un mínimo de cuatro *bytes*.

En cuanto a la codificación de valores enteros [1, 4], se asume una estrategia MSB (*Most Significant Byte*) y complemento a 2. Si bien la estrategia MSB no es la única existente (en contraste, por ejemplo con la LSB (*Least Significant Byte*)[2], se ha decidido no parametrizar este aspecto de la herramienta para no introducir mayor complejidad. De hecho, para facilitar la comprensión del estudiante, se puede configurar fácilmente C-Sim para que muestre los valores en decimal en lugar de hexadecimal.

C-Sim permte mostrar cualquier valor relacionado con las variables creadas fácilmente, utilizando el metacomando '?'. Así, "*? x*" y "*? *ptrX*" mostrarán el mismo resultado, 5, así como "*? &x*" y "*? ptrX*", que se corresponde con al dirección de memoria en la que se encuentra *x*. Es interesante reseñar que las variables se colocan aleatoriamente en cualquier posición de la memoria, por lo que, al igual que en cualquier programa real, no es posible predecir el lugar de la memoria en la que reside una variable.

```
int &refX = x
refX = 9
```

Con las dos líneas de código anteriores, se crea la referencia *refX*, que apunta a la variable *x*. Las referencias se tratan como punteros especiales que a) deben ser inicializados obligatoriamente, y b) no necesitan de sintaxis especial para acceder a la variable a la que apuntan. Tras las sentencias anteriores, *? x* devolverá 9 en la salida. De la misma forma, las siguientes sentencias cambiarán el valor de *x* a 6, y lo muestran por la salida.

```
*ptr = 5
? *ptr
```

Desgraciadamente, es común intentar realizar un manejo incorrecto de memoria a través de un puntero. Dado que un puntero sólo contiene la información de la dirección de memoria, y el número de *bytes* afectados, no es posible realizar ningún tipo de comprobación acerca de si existe una variable adecuada o no realmente apuntada. Por ejemplo, las siguientes instrucciones crean una variable de tipo carácter, y tratan de utilizar *ptr* para señalarla.

```
char ch = 'A'
ptrX = &ch
```

Al igual que si hubiera sido escrita en C o C++, la última sentencia no es aceptada. El tipo de la variable no coincide con el tipo del puntero, por lo que no se permite la asignación directamente. Sin embargo, en C y C++ siempre sería posible realizar la asignación [2, 5], utilizando un *cast* de la forma *ptr = (int *) ch*. Si bien los *cast* no están soportados en C-Sim, siempre es posible asignar directamente la posición de memoria en la que reside *ch* (este valor cambiará de una ejecución a otra).

```
ptrX = 90
? *ptrX
```

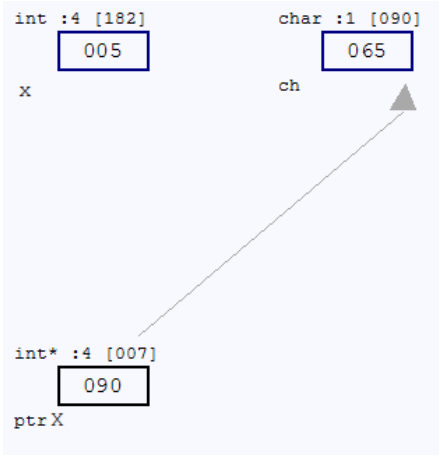

Figura 3: Diagramador mostrando un puntero a **int** señalando a una variable de tipo **char**.

El diagramador muestra ahora (véase la Figura 3), cómo *ptr* (con valor 90) apunta a *ch* (en la dirección 90), a pesar de que los tipos no coinciden. De hecho, tras la segunda sentencia, se mostrará por la salida estándar 0x41000000, ya que se toman (incorrectamente) el *byte* ocupado por *ch*, así como los siguientes tres *bytes*. Este valor, en realidad, depende de cómo se haya inicializado la memoria,. Por defecto se inicializa con ceros, pero es posible realizar una inicialización aleatoria, que probablemente será más realista, pero también, en un

[2]Esta cuestión es también conocida como *big endian* y *little endian*.

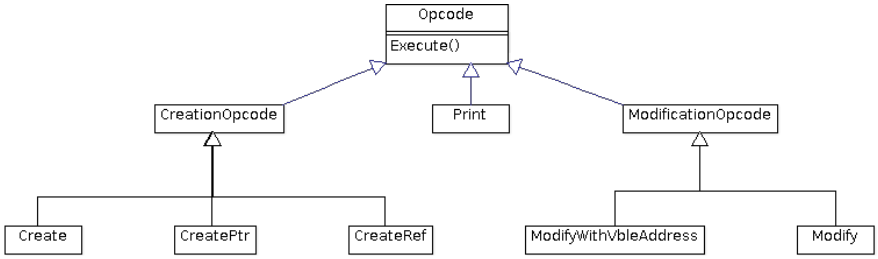

Figura 5: Conjunto de *opcodes* soportado por C-Sim.

principio, más confusa. Por otra parte, de esta forma, el alumno se dará cuenta de los posibles errores en tiempo de ejecución que puede causar el uso incorrecto de punteros

2.3. Diseño de C-Sim

C-Sim ha sido desarrollado como un intérprete con un entorno de ejecución integrado, debido a sus especiales necesidades de ser accesible y fácilmente manejable, de manera que favorezca la interactividad.

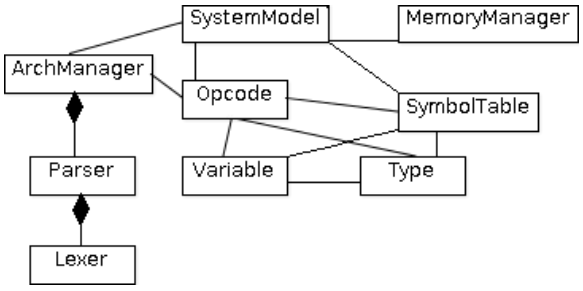

Figura 4: Estructura de clases de C-Sim.

El funcionamiento de C-Sim es muy sencillo (la arquitectura de clases puede verse en la Figura 4): la clase *facade* (fachada) **ArchManager** es el punto de entrada de instrucciones. Sendos objetos **Lexer** y **Parser** descomponen dichas instrucciones, generando los *opcodes* (Figura 5) correspondientes. Finalmente, dichos *opcodes* se ejecutan en secuencia, devolviendo una serie de variables que han sido modificadas o generando un error. En caso de éxito, se actualizan

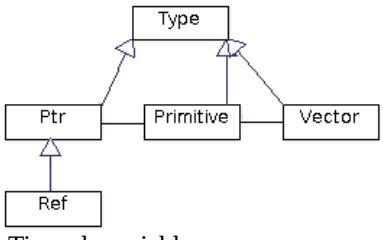

Figura 6: Tipos de variables.

entonces las posiciones de dichas variables en la representación de la memoria, a la vez que se actualiza también el árbol y el diagrama de variables. En caso de error, se informa del error al usuario, y la cadena de ejecución de *opcodes* se cancela completamente.

Los tipos de variables soportados están representados en el diagrama de la Figura 6. Se aprecia perfectamente cómo las referencias son efectivamente un tipo especial de punteros, mientras los tipos primitivos son utilizados por los punteros y los vectores, como tipos asociados.

C-Sim ha sido programado en C# sobre Mono (`http://mono-project.com/`), y ha sido ejecutado, demostrando una funcionalidad completa, en Ubuntu Linux 13.10, Windows 7, y Windows 8.

3. Trabajo relacionado

Si bien hay bastantes trabajos relacionados con la educación, o la asistencia a la educación en ingeniería informática, no existen demasiados trabajos relacionados, en cambio, con el tipo de herramienta presentado en este documento. Las herramientas relacionadas más extendidas son los tradicionales depuradores que acompañan a lenguajes de programación de nivel intermedio como C, las cuales sin embargo, no suelen ofrecer las facilidades que aporta C-Sim.

Python tutor (`http://pythontutor.com/`), es probablemente el recurso mejor relacionado, aunque en términos generales más ambicioso. El programa web permite mostrar la ejecución de un programa paso a paso, dibujando y actualizando diagramas sobre cómo se distribuyen los objetos en memoria. Esta aplicación es a la vez más y menos ambiciosa que C-Sim, pues permite simular la ejecución completa de un programa, pero a la vez las características de Python (`http://www.python.org/`), un lenguaje de programación en el que sólo existen referencias y variables primitivas, hacen que no tenga demasiado sentido profundizar en la diferencia entre referencias y punteros, accesos a memoria y obtención de direcciones de memoria, etc.

4. Conclusiones

En este artículo se ha presentado C-Sim como una herramienta de asistencia a la docencia en

programación de bajo nivel en C/C++. Esta herramienta se ha venido utilizando con éxito en los cursos 2012/13 y 2013/14 en las clases magistrales relacionadas con punteros, referencias y gestión de memoria.

La gran ventaja, es que permite observar las consecuencias de cualquier acceso a memoria de inmediato, permitiendo experimentar con los distintos operadores y punteros.

El trabajo futuro consistirá en añadir más funcionalidad a C-Sim, de tal manera que sea incluso capaz de ejecutar pequeñas funciones, y de esta manera poder ilustrar también las diferencias entre *Heap* y *Stack*.

Referencias

[1] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 4ª ed., 1992. ISBN 0-13-588187-0. ISBN 978-0321776419.

[2] Brian W. Kernighan, Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2ª ed., Abril de 1998. ISBN 978-0131103627.

[3] ISO. *ISO/IEC 14882:2011 Information technology -- Programming languages – C++* International Standards Organization. Septiembre de 2011.

[4] Paul Carter. *PC Assembly Language*. Lulu.com, 2009. ISBN 5-80031-684515.

[5] Stephen G. Kochan. Programming in C. Addison-Wesley.