Aprendiendo a Programar con la ayuda de un COAC#

Jakeline Marcos Abed, Tecnológico de Monterrey,
Campus Monterrey, Departamento de Ciencias Computacionales
jakeline@itesm.mx

Resumen

En la actualidad existen muchas herramientas que apoyan la comprensión de conceptos de programación. Muchas de ellas son herramientas que simulan un *debbuger*, otras muestran un diagrama de flujo dinámico pero sin visualizar el código. Algunas son aplicaciones *stand-alone*, mientras que otras son aplicaciones *web*. La intención de la herramienta que aquí se muestra es facilitar la comprensión de ciclos básicos, recorrido de *strings*, y conocimientos básicos sobre arreglos y matrices, para aquellos estudiantes sin experiencia previa en programación.

Aunque la herramienta sigue en mejora continua y se sigue analizando el impacto en la comprensión de los conceptos en los estudiantes, los resultados iniciales muestran que es una herramienta que favorece el aprendizaje de ciclos, *strings*, arreglos y matrices, en el aprendizaje de lenguajes de programación.

Palabras clave: programación, ciclos, estructuras de control

1. Introducción

Con la rápida evolución de la tecnología en los últimos años, la educación ha pasado de ser una disciplina puramente de enseñanza, a ser todo un ambiente de aprendizaje soportado por la tecnología (Yadin, 2011). El antiguo escenario donde el profesor impartía su clase y los estudiantes pasivamente escuchaban tratando de entender la explicación del profesor, ya no está vigente. El objetivo ahora es la exploración activa, la construcción dinámica de los conocimientos con apoyo en las tecnologías de información, y no solamente asistir a clase de forma pasiva (Norman & Spohrer, 1996).

Junto con la tecnología, las estrategias, técnicas y herramientas relacionadas con la enseñanza han evolucionado a lo largo del tiempo. Particularmente, la enseñanza de lenguajes de programación ha sido todo un reto, debido a la naturaleza abstracta de los lenguajes y desarrollo del pensamiento lógico del estudiante. En este sentido, el reto que supone la enseñanza de los lenguajes de programación es un problema y preocupación de muchos profesores que pertenecen al área de las Ciencias Computacionales.

2. ¿Qué se enseña en un curso de Programación y por qué es difícil?

De acuerdo a la taxonomía de Bloom (Bloom, 1956; Starr, Manaris, & Stalvey, 2008) en un curso de Fundamentos de Programación, se espera que un estudiante pueda:

- Recitar la definición y estructura de una sentencia de repetición o ciclo. (Nivel recuerdo)
- Explicar un "ciclo". (Nivel de comprensión).

- Rastrear código que contiene un "ciclo" (Nivel de aplicación).
- Explicar cómo una "sentencia de ciclo" se puede aplicar en la solución de un problema. (Nivel de análisis).
- Diseñar un "ciclo" desde cero para resolver un problema particular (Nivel de síntesis).

De todo lo anterior podemos decir que el logro de un nivel de síntesis del concepto de ciclos en un curso de Fundamentos de Programación, significa una profunda comprensión de las estructuras de control iterativos, por lo que el alumno tiene que ser capaz de escribir programas usando ciclos que realicen una tarea específica, creando soluciones completamente nuevas.
Por otro lado, según una encuesta realizada a 350 profesores de diferentes universidades de EE.UU. (Dale, 2006) en la que se les preguntó cuál es el tema más difícil para los estudiantes principiantes de la clase de Fundamentos de Programación (CS1), sus respuestas fueron de acuerdo a las siguientes cuatro categorías:

1. La resolución de problemas y el diseño
2. Estructuras de control, parámetros y recursividad
3. Orientación a objetos: herencia, polimorfismo, métodos, clases
4. Falta de madurez en los estudiantes y buenos hábitos de estudio

En la segunda categoría la enseñanza de las estructuras de control, tales como ciclos o ciclos con arreglos y matrices, definitivamente no es una tarea fácil. De hecho, en el desarrollo de habilidades de programación un factor determinante es el dominio de las estructuras de control, específicamente los de repetición (Milne & Rowe, 2002).
En este sentido, el uso de ciclos representa ciertas "dificultades" para los estudiantes que por primera vez están aprendiendo un lenguaje de programación. Para ellos, algunos errores comunes en la programación son:

- *Off-by-one*: Al escribir ciclos, se debe prestar especial atención a la especificación de la condición. Un error muy común es terminar el ciclo una pasada antes, o bien repetirla una vez más (Bell & Parr, 2010). Este problema puede presentarse por ejemplo cuando el estudiante utiliza "<=", cuando en realidad debería haber utilizado "<" en una comparación, o no toma en cuenta que un ciclo debe iniciar en cero en vez de uno.
- Ciclo infinito: Al construir la condición de un ciclo, es importante asegurarse de que no se ejecuta infinitamente (Overland, 2011).
- Índice fuera de rango: Un error común es confundir la longitud de un arreglo o un *string* con el rango de índices válidos. Este es el caso en el que el programa marca un error de ejecución de "Índice fuera de rango" (Bell & Parr, 2010).
- Falta de comprensión de la secuencia de: inicialización-condición-incremento. Un error común es confundir esta secuencia de pasos dentro de la instrucción *for*.

2.1 ¿Qué estamos haciendo?

Durante muchos años, en el departamento de Ciencias Computacionales hemos estado utilizando muchos lenguajes computacionales y excelentes herramientas para enseñar Fundamentos de Programación como: Alice, Raptor, Java, C ++ y C #. Sin embargo, un factor común es que es difícil para los estudiantes comprender las instrucciones de flujo de control, en particular los ciclos, no importa el paradigma o idioma (Dale, 2006; Kaczmarczyk, Petrick, East & German, 2010).

2.2 El diseño de una herramienta llamada COAC#

Con el propósito de ayudar a los estudiantes a aprender un lenguaje computacional, se diseñó una herramienta que pudiera simular la ejecución de un código. La idea era que este código pudiera ser manipulado por el usuario.

La aplicación se le llamó COAC# pues inicialmente fue desarrollada para C#, pero después se agregó la versión de C++ y JAVA.

El diseño de la aplicación está dividido en 4 partes (Fig. 1):

1. El código escrito en C#, C++ o JAVA
2. Los resultados de la ejecución del código
3. El diagrama de flujo animado, que permite configurar algunos valores propios del código
4. Y cómo van cambiando las variables en memoria

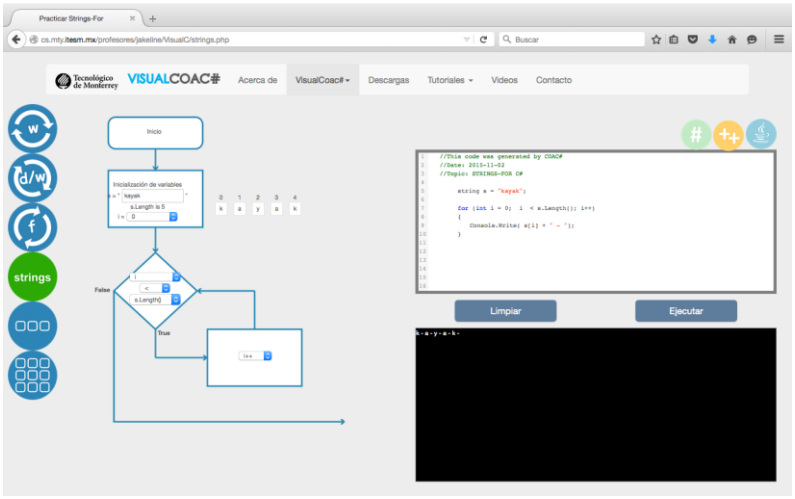

Figura 1

Algunas ventajas de esta herramienta son:

- Mejor visualización de los estatutos de repetición, al mostrar la ejecución en un diagrama de flujo animado.
- Mayor entendimiento de lo que sucede en memoria, al mostrar cómo van cambiando los valores de las variables.
- Posibilidad de establecer diferentes valores para las variables y condiciones.
- Visualización del código en C#, C++ y JAVA; con un botón se selecciona el lenguaje.
- Se puede instalar en Windows o utilizar la versión en Internet.
- Retroalimentación inmediata de errores, como: ciclo infinito, índice fuera de rango, etc.

2.3 Metodología

Para evaluar la herramienta propuesta, se invitó a 37 alumnos de la clase de Fundamentos de Programación a utilizarla, ya sea la versión en Windows o la nueva versión *web*. Los alumnos pertenecían a 2 grupos diferentes de segundo semestre, con el mismo profesor, de las carreras de Licenciado en Arte y Animación Digital e Ingeniería en Negocios y Tecnologías.

Además de que los alumnos utilizaron la herramienta, el profesor asignado a estos grupos la utilizó en su explicación de clase al iniciar cada uno de los siguientes temas: *while*, *do-while*, *for*, *strings*, arreglos y matrices. Dentro de su explicación, el profesor enfatizaba la secuencia lógica, el funcionamiento dinámico de los ciclos, el código mostrado en la ventana, y los diferentes casos de prueba que pudieran favorecer en la comprensión del concepto. Además de esto, el profesor mencionaba a los alumnos los principales "riesgos de error" al utilizar determinada secuencia de instrucciones.

Después de que los alumnos utilizaron la herramienta como autoestudio en el salón, o en sus tareas, se aplicó un examen de los temas: *while*, *do-while*, *for*, *strings*, arreglos y matrices. Dentro del examen, se incluyó un problema relacionado con cada uno de estos temas, teniendo como hipótesis que después de haber utilizado la herramienta varias veces, los estudiantes no deberían de caer en los errores asociados al problema.

3. Resultados

La siguiente tabla es una muestra de algunas preguntas que se hicieron en el examen. Además de estas preguntas, también se incluyeron otros ejercicios que implicaban desarrollo de código.

Los resultados obtenidos específicamente en estas preguntas, para este grupo de 37 alumnos fueron los siguientes:

Controlar error	**Ejemplo del código que se evaluó:**	**Alumnos que Sí contestaron correctamente** ✓	**Alumnos que NO contestaron correctamente** ✗
Off-by-one error, Ciclo infinito y correcta construcción del ciclo	1.- Manejo de Strings. Desarrollar el código necesario para mostrar en pantalla las letras de una palabra, en forma intercalada, pero recorriéndola al revés. Ejemplo: la palabra "maestro" se desplegaría: "o t e m"	**28**	**9**

Índice fuera de rango	2.- ¿Cuáles de las siguientes secuencias de estatutos marcan error al tratar de sumar todos los elementos del arreglo *arr* previamente definido e inicializado?
	a) int suma = 0;
	for (int j = arr.Length; j >= 0; j --)
	suma = suma + arr[j];
	b) int suma = 0;
	for (int j = arr.Length-1; j >= 0; j++)
	suma = suma + arr[j];
		30	**7**

	c) int suma = 0;
	for (int j = 0; j < arr.Length; j=j+1)
	suma = suma + arr[j];
	
	d) int suma = 0;
	for (int j = 0; j <= arr.Length-1; j++)
	suma = suma + arr[j] ;		

Falta de comprensión de la secuencia de: inicialización-condición-incremento	3.- Dado el siguiente código:
	
	double suma = 3;
	for(int i = 7; i > 10; i = i - 2){
	suma = suma + i;
	}
	¿Cuál es el valor final de la variable suma?
		34	**3**

Falta de comprensión al construir la condición del ciclo	4.- Para producir exactamente la salida "3 6 9 12 15", ¿Cuál debe ser la condición del *loop* para el siguiente ciclo?
	
	n = 0;
	do {
	n = n + 3;
	Console.Write(n + " ");
	} while(_____);
		26	**11**

Tabla 1

4. Discusión

Al utilizar el COAC# en el curso de Fundamentos de Programación, se observó que el uso de la herramienta durante la clase sí mejora la visualización y comprensión de algunos conceptos. De hecho, se vio reducido el número de estudiantes que caen en algunos errores, principalmente en los problemas de "recorrido" de código, como el problema 3.

Sin embargo, a la hora de codificar o escribir soluciones completas, aún se observan deficiencias relacionadas con la lógica y la sintaxis, como el problema 1. Esto muchas veces tiene que ver con la falta de madurez para resolver problemas razonados, o para establecer relaciones o condiciones simples de menor, mayor o igual, por ejemplo el problema 4.

Por otro lado, se ha visto que es una gran ventaja el tener la nueva versión de la herramienta corriendo como aplicación *web,* ya que los alumnos la pueden utilizar en cualquier momento y en cualquier plataforma. De hecho, de los 37 alumnos, 29 de ellos dijeron que utilizaron la herramienta para practicar para el examen, lo cual repercutió favorablemente en algunos temas.

5. Conclusiones

Definitivamente el tema de ciclos, *strings* y arreglos, representa una gran dificultad para los alumnos que nunca han llevado clase de programación. Aunque no podemos pensar en la herramienta COAC# como una bala de plata, que resuelve todo y garantiza el total entendimiento de los temas, sí se puede considerar de gran ayuda tanto para los alumnos como para el profesor. De hecho, para los estudiantes es importante tener una herramienta práctica que les permita visualizar el código y "averiguar" el "qué pasa si...", de manera que puedan equivocarse, pero también aprender de sus errores.

Uno de los factores clave de esta herramienta es la visualización de conceptos, al expresar de forma gráfica el funcionamiento de algunos estatutos, que de otra forma serían muy abstractos para el alumno. La comprensión de estos conceptos está relacionada directamente con la comprensión del

por qué de los errores. De esta forma el estudiante aprende a corregir sus errores y, lo más importante, aprende a no caer en ellos, lo cual es una parte crítica a la hora de programar.

Para aumentar las posibilidades de éxito en la materia Fundamentos de Programación, la recomendación final es permitir que los alumnos practiquen el tiempo suficiente, no solo con la herramienta, sino en el desarrollo completo de soluciones y teniendo la herramienta COAC# como un soporte al aprendizaje.

4. Referencias

- Bell & Parr (2010). *C# for Students*. Revised edition. Pearson Education.
- Bloom, B. S. (1956). Taxonomy of educational objectives: Handbook 1: Cognitive domain. Longmans, Green and Company, New York.
- D. A. Norman and J. C. Spohrer (1996), Learner-centered education. *Communications of the ACM, 39* (4).
- Dale, N. (2006). Most difficult topics in CS1: results of an online survey of educators. *SIGCSE Bulletin, 38* (2).
- Kaczmarczyk, L., Petrick, E., East, P. and German, G. (2010). Identifying Student Misconceptions of Programming. SIGCSE (Milwaukee, Wisconsin, USA, March 10-13, 2010).
- Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming – Views of Students and Tutors. *Education and Information Technologies, 7*.
- Overland, B. (2011). *C++ Without Fear: A Beginner's Guide That Makes You Feel Smart*. Second Edition. Prentice Hall.
- Starr, C., Manaris, B., Stalvey, R. (2008). Bloom's Taxonomy Revisited: Specifying Assessable Learning Objectives in Computer Science. SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.
- Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads, 2* (4).