Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

PROTOTIPO DE APLICACIÓN WEB PARA CONSTRUCCIÓN DE CÓDIGO BÁSICO EN LENGUAJE C A TRAVÉS DEL USO DE UNA INTERFAZ GRÁFICA A BASE DE BLOQUES

Axel Ernesto Moreno Cervantes
Instituto Politécnico Nacional
amorenoc@ipn.mx

Verónica Agustín Domínguez
Instituto Politécnico Nacional
vagustin@ipn.mx

Benjamín Cruz Torres
Instituto Politécnico Nacional
bcruzt@ipn.mx

Abstract

En este trabajo se presenta el desarrollo de un prototipo de aplicación web que utiliza una interfaz gráfica a base de bloques para construir código en el lenguaje de programación C, facilitando así el proceso de codificación y de aprendizaje de la lógica de programación de lenguaje C y por lo tanto, pueda utilizarse como una herramienta de apoyo para aprender o enseñar dicho lenguaje de programación. El desarrollo tomó como guía la metodología de desarrollo de software basada en prototipos y para su programación se utilizaron las siguientes tecnologías: Blockly, Typescript y HTML5 para la interfaz gráfica y uso de bloques, node js como entorno de ejecución, NPM como gestor de dependencias, webpack como gestor de recursos, así como para la compilación de la aplicación y Azure para el despliegue de la aplicación.

Palabras clave: Lenguaje C, Aplicación web, Codificación basada en bloques, Herramienta de apoyo a la enseñanza de programación.

Las dificultades presentes en el aprendizaje de la programación consisten algunos factores tales como, dificultad con el pensamiento lógico y algorítmico, sobrecarga de conceptos nuevos, problemas para comprender el flujo de ejecución, falta de paciencia y frustración con la depuración, entre otras. Pero sumado a eso, los estudiantes también batallan en tener que aprender la sintaxis de programación, los cuales, suelen ser difíciles de recordar y entender (Cheah, 2020).

También, algunos estudios académicos y de investigación pedagógica, han señalado errores de sintaxis cometidos por estudiantes principiantes relacionados con olvidar colocar puntos y comas y llaves, confundir el operador de asignación (=) con el de comparación (==), entre otros (Kadar, Wahab, Othman, & Shamsuddin, 2021).

Como último punto relacionado a las problemáticas asociadas a la comprensión de la sintaxis en el aprendizaje de un lenguaje de

programación, en una encuesta realizada a 57 alumnos quienes habían terminado un curso de programación en la Universidad IQRA en Karachi, los resultados mostraron que el mayor reto al que los estudiantes se enfrentaron mientras programaban fue del de entender la sintaxis de programación (Islam, 2019).

Una herramienta que puede facilitar la codificación de un programa y subsanar los problemas mencionados es la codificación basada en bloques, en donde las instrucciones, sentencias y en general, la sintaxis, se encuentran en forma de bloques con una forma similar al de un rompecabezas y que pueden arrastrarse y soltarse para unirse con otros bloques (Figura 1). Entre una de las ventajas que ofrecen los entornos de programación basados en bloques, es la de evitar errores de sintaxis mientras se conserva la práctica de crear programas ensamblando sentencias una por una, esto gracias a que cada instrucción se encuentra en un bloque que se puede arrastrar y soltar para unirse con otros bloques de instrucciones ahorrándose la necesidad de escribirlas. Tan solo el formato visual y la mitigación de errores de sintaxis son un aspecto clave para ayudar a los principiantes a poder llevar a cabo la programación de un programa (Weintrop, 2019).

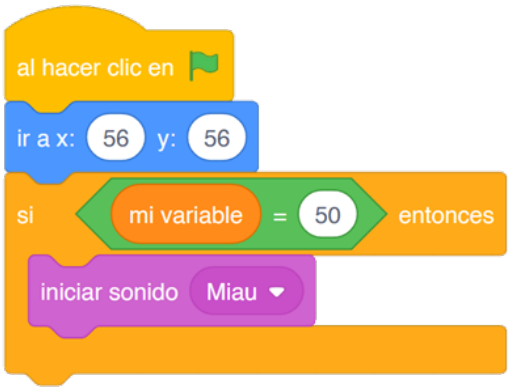

Figura 1. Bloques de código del lenguaje de programación Scratch, que utiliza una interfaz de codificación basada en bloques.

El lenguaje de programación C, ha sido uno de los lenguajes de programación más utilizados desde el año 1970, ya que, al tratarse de un lenguaje de propósito general, es utilizado en la creación de sistemas operativos, compiladores, sistemas embebidos y otros tipos de software de bajo nivel (Chavez, Gitalan, & Lincopinis, 2021). También y gracias a que a que es un lenguaje de nivel medio por tener elementos de lenguajes de alto nivel y la flexibilidad de un lenguaje a bajo nivel como ensamblador (Schildt, Hernández, & Vaquero, 2003), permite comprender mejor el manejo de recursos de la computadora por parte de un programa. Es por eso, que suele ser el lenguaje de programación que se utiliza en planes de estudio de introducción a la programación, como lo es en el caso de la Escuela Superior de Cómputo del Instituto Politécnico Nacional.

Es por eso, que se propuso desarrollar un prototipo aplicación web que permita construir código en lenguaje C utilizando una interfaz gráfica a base de bloques en la que se pueda generar código en lenguaje C a partir de un código en forma de bloques.

I. Metodología

1) Dependencias y tecnologías empleadas.

Con el objetivo de facilitar la implementación de la interfaz gráfica a base de bloques y no haber tenido que construir una desde cero, se utilizó la biblioteca de Blockly, que permite "…agregar un editor personalizable a tu app que presenta conceptos de programación como bloques entrelazados (Google, s.f.)". Y que además, "Produce código limpio en el lenguaje deseado y permite

bloques personalizados adaptados a tu aplicación (Google, s.f.) ".

Además de las tecnologías básicas para la web (HTML5, CSS y JavaScript), otras tecnologías importantes utilizadas para el desarrollo del proyecto fueron:

1) TypeScript: lenguaje de programación que incluye todas las características de JavaScript, pero con la particularidad de agregar tipado estático (w3schools, s.f.). Fue utilizado como alternativa a JavaScript para la programación en la web, ya que, al estar tipado, se mejora la escalabilidad, depuración y mejor manejo del código.
2) Node.js: usado como entorno de ejecución para permitir que la aplicación se ejecute en el lado del servidor. Fue usado gracias a que posee ventajas para el desarrollo de aplicaciones web, tales como un alto rendimiento, desarrollo de aplicaciones en tiempo real, soporte multiplataforma, etc. (Le, s.f.).

3) NPM (Node Package Modules): es el es el administrador de paquetes de Node.js, que permite a los desarrolladores de JavaScript trabajar con dependencias, siendo estas bibliotecas que se usan para desarrollar un programa sin la necesidad de hacer todo el trabajo desde cero (Zúñiga, s.f.)

4) Webpack: Es un empaquetador de módulos estáticos utilizado para aplicaciones en JavaScript (Concepts, s.f.). Se utilizó para administrar el código de la aplicación en módulos, y, además, para optimizar y minimizar el código para el despliegue.

5) Azure: Es un servicio de computación en la nube que ofrece más de 200 productos y servicios en la nube (Microsoft, s.f.). Este servicio fue utilizado para realizar el despliegue de la aplicación y que pueda ser accedido de manera remota a través web.

6) GitHub Actions: "plataforma de integración y despliegue continuos (IC/DC) que te permite automatizar tu mapa de compilación, pruebas y despliegue" (GitHub, s.f.). Fue empleado junto con Azure para realizar el proceso de despliegue desde el repositorio del código fuente que se encuentra alojado en la plataforma GitHub.

Finalmente, se utilizaron algunas dependencias tanto para agregar componentes y funcionalidades a la aplicación, así como para Blockly:

- Toastify-js: biblioteca ligera escrita en JavaScript que permite añadir notificaciones tipo "Toast".

- Sweetalert2: permite añadir cuadros emergentes responsivos, personalizables y accesibles para JavaScript.

- Workspace-multiselect: "permite arrastrar, seleccionar y manipular múltiples bloques en el espacio de trabajo". Esta dependencia fue utilizada porque estas acciones no están incorporadas por defecto en la biblioteca Blockly.

2) Metodología e implementación

En cuanto al modelo de la metodología para llevar a cabo el desarrollo del proyecto, se utilizó una de proceso evolutivo, que se caracteriza por ser iterativo y permitir desarrollar versiones cada vez más completas del software (Pressman, 2013). De este

modelo, el paradigma empleado para el desarrollo del proyecto fue el de "hacer prototipos". En este paradigma, se comienza con una comunicación con otros participantes del equipo para definir los objetivos generales del software y los requerimientos que se sabe necesita el proyecto, posteriormente, se planea una iteración rápida para hacer el prototipo y se lleva a cabo un diseño rápido en la que se toma en cuenta principalmente las características visibles para los usuarios finales (como por ejemplo el diseño de la interfaz de usuario), este diseño rápido finalmente conduce al desarrollo de un prototipo que posteriormente será evaluado por los participantes quienes darán retroalimentación para mejorar los requerimientos. Cuando el prototipo es afinado para satisfacer las necesidades de distintos participantes, se produce una iteración, con la que se puede entender mejor lo que se necesita hacer (Pressman, 2013).

El desarrollo fue divido en tres prototipos, las cuales se abordan a continuación:

A. Prototipo I: creación de bloques.

En este prototipo, se desarrolló no solamente la estructura y diseño de los bloques, sino también las funciones dinámicas y responsivas de los bloques necesarias para su uso en la codificación.

Antes de desarrollar cada bloque, se pensó en el diseño de la caja de herramientas (componente en donde el usuario puede acceder a los bloques para construir su programa, Figura 3) para categorizar los bloques, y que se hizo con respecto a sus funcionalidades en común, quedando de la siguiente manera:

1. Funciones: bloques para definir, utilizar y en general relacionados al uso de funciones en C.

2. Variables: bloques relacionados al uso de variables y otros elementos (apuntadores, instancias de estructuras y arreglos). Se incorporaron las tres acciones más importantes en el lenguaje (declaración, uso con respecto al alcance del elemento declarado y modificación).

3. Valores: bloques para utilizar valores constantes (valores numéricos, valor NULL para los apuntadores y un bloque para inicializar elementos de un arreglo).

4. Condición: para utilizar controles de flujos condicionales. Los implementados fueron, la condición if – else para el caso de una sola condición en particular, y la sentencia switch para casos de condiciones múltiples.

5. Matemáticas: para emplear la sintaxis de operaciones básicas del lenguaje C (suma, resta, multiplicación, división y módulo), al igual que para utilizar las funciones y constantes matemáticas básicas y más utilizadas procedentes de la biblioteca math.h.

6. Lógica: para realizar comparaciones entre dos valores y operaciones lógicas.

7. Bucle: sentencias básicas relacionados al uso de bucles (while, do-while, break, continue y for simplificado para iterar con base a una variable al que se le realiza una operación con un valor determinado).

8. Impresión: para imprimir texto y variables en salida estándar (consola).

9. Lectura: para leer una variable a la vez desde la entrada estándar o consola.

10. Estructura: definir estructuras y acceder a los miembros de instancias declaradas.

11. Apuntadores: para obtener el dato apuntado por un apuntador (operador de indirección).

12. Memoria: bloques relacionados a funciones para usar y manipular memoria, las que fueron implementados son: obtener la dirección de memoria de una variable o elemento (operador &), reserva de memoria haciendo uso de la función malloc y liberación de memoria.

Figura 2. Caja de herramientas de la aplicación.

Para la implementación de los bloques, se utilizaron los dos componentes básicos y principales de un bloque en la biblioteca, los cuales permiten parametrizar al bloque de la forma que el usuario desee y así construir el código de su programa a sus necesidades. Estos componentes son:

1. Campos: son espacios en donde el usuario puede introducir un valor o seleccionar alguno de un conjunto; además también se utiliza como elementos de un bloque que muestran información sobre el mismo, como texto, imágenes, etc. (ver Figura 3).

2. Entradas: son conexiones que tiene un bloque en la que se puede insertar otros bloques (ver Figura 3).

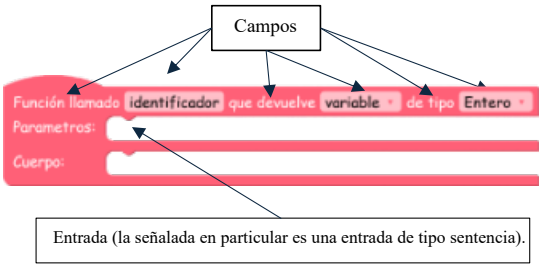

Figura 3. Campos y entradas de un bloque de la aplicación.

Para el desarrollo de las funciones dinámicas y responsivas, se utilizaron eventos provenientes de la biblioteca que se ejecutan cuando se realiza alguna acción en el espacio de trabajo (área en donde se encuentran todos los bloques para construir el código, ver Figura 4).

También, se añadieron componentes y módulos en el código de la aplicación encargados de controlar las acciones y propiedades de los bloques relacionados a ellas, como, por ejemplo, el control de definición de estructuras hechas por el usuario que posteriormente se mostraran dinámicamente en los campos de los bloques que la pueden utilizar.

Humanidades, Tecnología y Ciencia,
del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

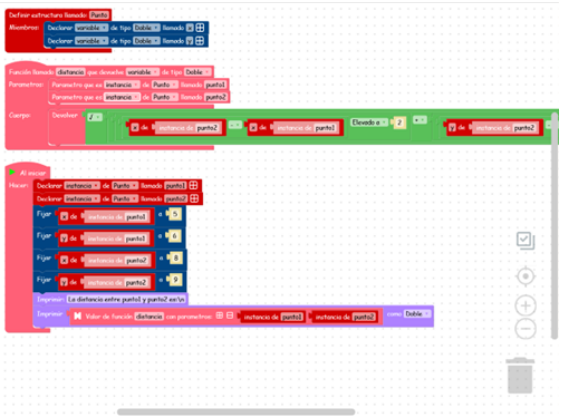

Figura 4. Espacio de trabajo con bloques de la aplicación.

Durante el periodo final del desarrollo de este prototipo, se realizaron pruebas para verificar el correcto funcionamiento de cada uno de los bloques desarrollados para corregir errores de programación presentados.

B. Prototipo II: generador de código.

Primero, a cada bloque se le asignó un generador de código con base a los valores que contiene sus entradas y campos, procurando, que la implementación se haga correctamente para adecuarse con el mecanismo de generación de código por defecto con la que funciona el componente de generador de código que tiene la biblioteca. También, al código generado del bloque, se le asigno un color análogo al bloque que lo genera, para que el usuario pueda identificarlo y asociarlos más rápidamente (ver Figura 5).

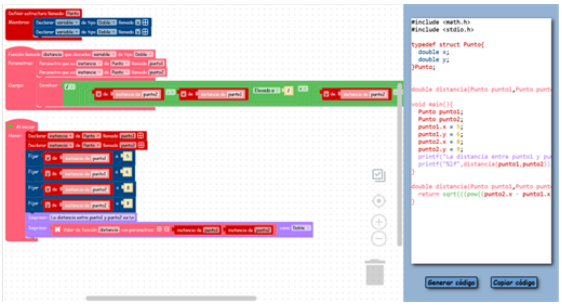

Figura 5. Código generado con respecto a los bloques del espacio de trabajo de la aplicación.

Posteriormente, se implementó un mecanismo de generación de código para que en las entradas de aquellos bloques donde se pueden colocar más bloques de instrucciones (entrada de sentencia, ver Figura 3) el código generado siguiese las reglas de sintaxis de C, principalmente, que al final de cada instrucción o sentencia se debe colocar un punto y coma, aunque no necesario en sentencias o instrucciones que tengan un cuerpo, como lo son las sentencias if – else, switch, while, etc.

Después, se modificó el mecanismo de generación de código para todo el espacio de trabajo, para que el código final se generara con el siguiente orden (con el objetivo de generar un código estructurado a las buenas prácticas de programación en el lenguaje):

1. Definiciones de estructuras (implementados con typedef).

2. Prototipos de función.

3. Función main.

4. Cuerpos de función.

Finalmente, se realizaron pruebas de la generación de código para corroborar que el código no solamente estuviese absuelto de errores sintácticos, sino también, fuese

congruente a los valores y entradas de cada bloque, al igual que fuese preprocesado correctamente en función a las variables y valores internos en el código del bloque con respecto a un campo seleccionado.

C. Prototipo III: Integración de funcionalidades y pruebas.

En este prototipo, se implementó y probó el sistema de guardado y cargado con las funciones de serialización y deserialización que ofrece la biblioteca. En el sistema de guardado también se incluyeron aquellos datos definidos por el usuario, que en este caso fueron las funciones y las estructuras.

Posteriormente, Webpack fue configurado para el despliegue de la aplicación. Una vez desplegada, se realizaron pruebas para verificar el desempeño de la aplicación ya desplegada y corroborar si no se necesitaba modificar las configuraciones de compilación para el despliegue.

Finalmente, como pruebas generales de la aplicación, se construyeron algunos programas utilizando la aplicación para analizar su funcionamiento en posibles usos que el usuario podría darle y de esa forma corroborar si hay errores de programación presentes o detalles con respecto al diseño y funcionamiento de sus componentes, al igual que su desempeño general al utilizar cierta cantidad de bloques.

II. Resultados

En el producto final, se observaron algunos pequeños errores de programación en el funcionamiento responsivo de los bloques, sobre todo, relacionados con la actualización de campos de lista desplegables a la hora de mostrar elementos dinámicamente en función

Sin embargo, al realizar las pruebas generales, se verificó que, con los bloques implementados en la aplicación, se puede realizar una gran variedad de programas básicos, e inclusive, se pueden realizar ciertas cosas más de lo previsto, y que son suficientes para aprender lo más básico del lenguaje en cuanto sintaxis, propiedades más sobresalientes y paradigma de programación.

III. Conclusiones

Durante el desarrollo del proyecto, hubo ciertas complicaciones en la implementación de ciertas funcionalidades en la forma deseada debido a que el funcionamiento por defecto de algunas características de la biblioteca Blockly no iba acorde y/o no permitían realizar lo que se buscaba hacer. Estos problemas tuvieron que ser subsanados como ya se había aludido anteriormente, implementando módulos externos o modificando clases, nombres de espacios y funciones de la biblioteca.

Uno de los mayores retos para el desarrollo de este proyecto, fue el de diseñar la forma y estructura de los bloques para adaptarse a la sintaxis del lenguaje C, durante el desarrollo del proyecto hubo muchas propuestas y cambios en los bloques debido a que cuando se pensaba un diseño que buscaba simplificar el uso del bloque y de la codificación en general, se dejaba de lado o se limitaba la capacidad de hacer (o implementar a futuro) ciertas funcionalidades que en el lenguaje si se podían hacer sin caer en la necesidad añadirle demasiados componentes y entradas a su diseño y estructura.

Finalmente, es importante mencionar sobre algunas limitaciones observados en la codificación por bloques en el uso de la sintaxis del lenguaje C, ya que, al estar basado en texto, se utilizan símbolos y caracteres específicos en un determinado orden para

delimitarlo. Esto hace que, por ejemplo, no se pueda nombrar una variable iniciando con un número, o que no se puedan utilizar caracteres que no sean alfanuméricos ya que, en el caso de este último, estos símbolos están reservados para delimitar la estructura sintáctica del lenguaje; o que el primer caso suceda, ya que el compilador piensa que se está escribiendo una expresión en vez de un identificador o nombre. Sin embargo, este tipo de limitaciones no deberían estar presentes en un lenguaje de programación cuya codificación está basada en bloques, puesto a que su estructura sintáctica está delimitada por las conexiones, campos y entradas del bloque y no por caracteres y símbolos.

Referencias

Chavez, B. B., Gitalan, T. A., & Lincopinis, D. R. (2021). C Programming Language: A Review. *Journal of Universal Computer Science*, 1-5. doi:10.3897/jucs

Cheah, C. S. (2020). actores que contribuyen a las dificultades en la enseñanza y el aprendizaje de la programación informática: Una revisión bibliográfica. *Tecnología Educativa Contemporánea, 12*(2), 272. doi:https://doi.org/10.30935/cedtech/8247

Concepts. (s.f.). Obtenido de Webpack: https://webpack.js.org/concepts/

GitHub. (s.f.). *Entender las GitHub Actions*. Obtenido de Documentación de GitHub: https://docs.github.com/es/actions/get-started/understand-github-actions

Google. (s.f.). *Codificación potente , bloque por bloque*. Obtenido de https://developers.google.com/blockly?hl=es-419

Islam, N. (2019). A Study of Difficulties of Students in Learning Programming. *Journal of Education & Social Sciences, 7*(2), 38-46. doi:https://doi.org/10.20547/jess0721907203

Kadar, R., Wahab, N. A., Othman, J., & Shamsuddin, M. (2021). A Study of Difficulties in Teaching and Learning Programming: A Systematic Literature Review. *International Journal of Academic Research in Progressive Education and Development, 10*(3), 591-605. doi:http://dx.doi.org/10.6007/IJARPED/v10-i3/11100

Le, V. (s.f.). *Top Node JS Advantages That Set It Apart from Other Runtime Environments*. Obtenido de https://www.orientsoftware.com/blog/node-js-advantages/

Microsoft. (s.f.). *Qué es Azure: Servicios en la nube de Microsoft*. Obtenido de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-azure

Pressman, R. S. (2013). *Ingeniería de software: un enfoque práctico* (7a ed.). Madrid: McGraw-Hill Education.

Schildt, H., Hernández, L., & Vaquero, A. (2003). *C : manual de referencia* (4a ed.). Madrid: McGraw Hill.

w3schools. (s.f.). *TypeScript Introduction*. Obtenido de https://www.w3schools.com/typescript/typescript_intro.php

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, 22-25. doi:https://doi.org/10.1145/334122

Zúñiga, F. (s.f.). *¿Qué es NPM? JavaScript para principiantes*. Obtenido de https://www.arsys.es/blog/que-es-npm-javascript-para-principiantes