**Primer Autor, Segundo Autor, Tercer Autor**

**No. de Equipo Trabajo: {*Número de Equipo de trabajo*}**

# [[1]](#footnote-1) INTRODUCCIÓN

Este documento consiste en una descripción detallada del desarrollo de un IDE (Entorno de desarrollo integrado) para el lenguaje de programación Brainfuck++.

# DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Brainfuck es un lenguaje de programación esotérico creado en 1993 por Urban Müller [2], el cual contiene solo 8 comandos, de los cuales 2 son para la salida y entrada de datos. Al igual que muchos otros lenguajes de programación, este lenguaje tiene diferentes extensiones, como lo es el caso de Brainfuck++, una extensión hecha por la página web *CodeAbbey* [1].

Hoy día, la mayoría de los lenguajes que se usan, como Python, C++ o JAVA, poseen, entre muchas cosas, una sintaxis más legible y estructurada para hacer más cosas con menos líneas de código. Sin embargo, al Brainfuck++ ser un lenguaje esotérico, desafía y reta al programador a ser más ingenioso y eficiente, desarrollando un mejor pensamiento lógico-matemático, que en un futuro facilita el aprendizaje de lenguajes con más *“comodidades”*.

Es por esta razón que como grupo hemos decidido desarrollar un IDE para Brainfuck++, que básicamente permita escribir y compilar código.

# usuarios DEL PRODUCTO DE SOFTWARE

Este software está diseñado para dos tipos de usuarios.

Primero: los estudiantes que están cursando cursos como *“Programación de computadores”* y no tienen conocimientos previos de programación, debido a que el funcionamiento de este lenguaje de programación es muy similar al de una máquina de Turing.

Segundo: Los usuarios con conocimientos previos de programación que quieran retarse a sí mismos con este lenguaje y de paso mejorar sus habilidades de razonamiento lógico-matemático.

# REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Los requerimientos funcionales de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones.

Entre los posibles requerimientos funcionales de un sistema, se incluyen:

* Descripciones de los datos a ser ingresados en el sistema.
* Descripciones de las operaciones a ser realizadas en cada pantalla que se presenta.
* Descripción de los flujos de trabajo realizados por el sistema.
* Descripción de los reportes del sistema y otras salidas.
* Definición de quiénes pueden ingresar datos en el sistema.

Título del Proyecto a Realizar

De esta manera, se deben describir las interacciones que tendrán los usuarios con el software.

Cada funcionalidad se debe especificar así:

## Nombre de la funcionalidad

En el título de la funcionalidad, se recomienda utilizar nombres muy descriptivos para cada funcionalidad. No limitarse a nombrarlas “Funcionalidad 1”, en cambio usar por ejemplo: “Autorización de pedido de compra”.

## Descripción: Descripción breve de la funcionalidad.

* *Acciones iniciadoras y comportamiento esperado*:

Secuencia de acciones del usuario y respuestas esperadas del programa para esta funcionalidad.

*Requerimientos funcionales*:

Lista detallada de los requerimientos funcionales asociados a esta funcionalidad.

Para cada requerimiento funcional se establece cómo debe mostrarse el software y cuáles comportamientos debe desempeñar para que el usuario pueda realizar la función que necesita.

Es recomendable prever y describir cómo debe responder el software ante condiciones de error y entradas de datos inválidas.

Las funcionalidades mínimas sobre los datos que se manejen deben prever operaciones de:

* Creación
* Actualización
* Eliminación
* Consulta total de los datos
* Búsqueda parcial de datos
* Ordenamiento
* Almacenamiento

Aunque en otros cursos se estudian estrategias de organización y almacenamiento, en este curso el almacenamiento se requiere principalmente para facilitar las pruebas del prototipo de software. También, para facilitar su implementación, se deja abierta la opción a que se apoyen en el uso de sistemas manejadores de bases de datos, o se haga almacenamiento por archivo, de objetos u otra estrategia que les convenga, siempre que se garantice la implementación y uso de las estructuras de datos vistas en clase en memoria.

IMPORTANTE: En cada una de las entregas para reportar el avance en el desarrollo del proyecto, se especificarán las funcionalidades mínimas y las estructuras de datos mínimas requeridas que se deben implementar. También, se debe presentar un análisis (especialmente, comparativo) breve de la eficiencia de las estructuras de datos usadas.

NOTA: En el siguiente enlace web (URL) puede encontrar una explicación de cómo diferenciar Requisitos Funcionales de los No Funcionales:

<https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

# DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

La interfaz de usuario se enfatiza en en el uso de tres paneles dentro de una ventana, el primer panel se encuentra dedicado a la programación en sintaxis de Brainfuck++, el segundo a las entradas y el tercero a la salida del programa.

La ventana tambien contiene una cinta de opciones, en la cual se podrán ejecutar distintas acciones como abrir, guardar, cerrar, deshacer, rehacer y correr. Está interfaz podrá cambiar de acuerdo a las nuevas implementaciones del software.

Primer Mockup:

Captura de pantalla de un celular

Descripción generada automáticamente

Segundo Mockup:

Captura de pantalla de un celular

Descripción generada automáticamente

Versión Actual:

Captura de pantalla de un celular

Descripción generada automáticamente

# Entornos de desarrollo y de operación

El software se está desarrollando en el SO windows 10 64 bits, en un procesador intel i5 6200U @ 2.3 GHz y una memoria ram de 4 GB.

Las aplicaciones creadas en java pueden ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora, por lo que nuestra aplicación podrá ejecutarse sin problemas en los sistemas operativos más comunes (Windows, MacOS, Linux).

# PROTOTIPO DE SOFTWARE INICIAL

Para esta entrega de avance en el desarrollo del proyecto, se debe realizar una primera versión de un prototipo de software funcional de acuerdo con los requisitos que se explican a continuación.

Primero, el software desarrollado se debe registrar en un repositorio de software Github[[2]](#footnote-2). Para facilitar el uso de esta plataforma, se sugiere estudiar el tutorial disponible en:

<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>

Se debe organizar el software en el repositorio de una manera estructurada como se ilustra en la Figura 1:



Figura 1. Estructura del repositorio requerida durante el proyecto.

Tenga en cuenta que por cada entrega se debe generar en el repositorio, una rama diferente.

Adicionalmente, para mantener una versión gráfica de desarrollo del repositorio, se podrá apoyar mediante una herramienta como Sourcetree, disponible en el siguiente URL:

<https://www.sourcetreeapp.com/>

Segundo, en este prototipo se deben implementar por lo menos dos instancias de la estructura de datos tipo **Lista implementada como arreglos**. Para aclarar, aquí Lista se refiere a una estructura de datos lineal, o contenedor de datos en la cual los datos se organizan de una manera lineal (como en una hilera) de tal manera que cada dato puede tener a lo sumo un antecesor y a lo sumo un sucesor.

Tercero, para estas listas implementadas como arreglos se deben soportar por lo menos las siguientes operaciones funcionales:

* Creación de la lista
* Inserción de un solo dato
* Actualización de un solo dato
* Eliminación de un solo datos
* Búsqueda de un solo dato
* Consulta de todos los datos
* Almacenamiento de los datos

Para facilitar su implementación, se deja abierta la opción a que se apoyen en el uso de sistemas manejadores de bases de datos, o se haga almacenamiento por archivo, de objetos u otra estrategia que les convenga, siempre que se garantice la implementación y uso de las estructuras de datos almacenadas en memoria dinámica vistas en clase.

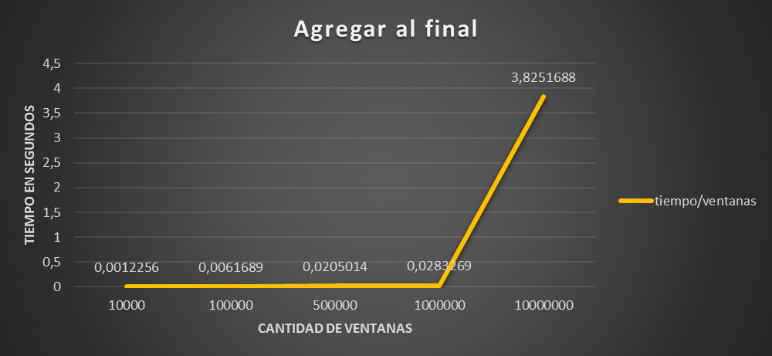
# PRUEBAS DEL PROTOTIPO

A continuación, se presentan las gráficas de las pruebas que se le hicieron a las diferentes funcionalidades del IDE.

***Ventanas de programación:*** Para las ventanas de programación se hicieron pruebas dependiendo de la cantidad de ventanas. A continuación, se muestran las tablas de comparación del tiempo de ejecución respecto a la cantidad de ventanas y sus respectivas gráficas.

*Agregar ventana al final de la lista:*

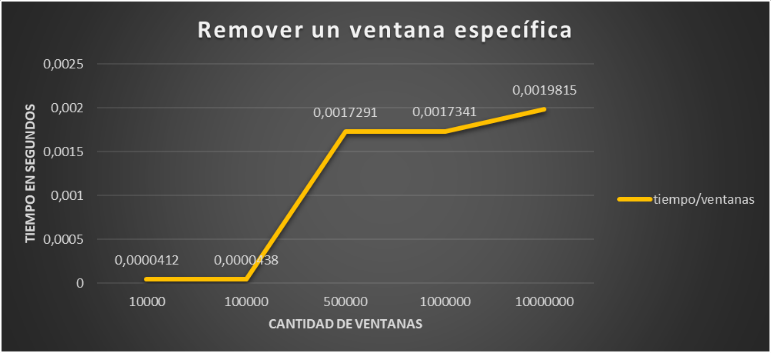
|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de ventanas | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 1225600ns | 1,2256ms | 0,0012256s |
| 100000 | 6168900ns | 6,1689ms | 0,0061689s |
| 500000 | 20501400ns | 20,5014ms | 0,0205014s |
| 1000000 | 28326900ns | 28,3269ms | 0,0283269s |
| 10000000 | 3825168800ns | 3825,1688ms | 3,8251688s |



*Complejidad: La complejidad es O(1) ya que las ventanas se agregan al final de una lista enlazada con un apuntador al final de la lista.*

*Remover ventana en una posición específica de la lista*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de ventanas | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 41200ns | 0,0412ms | 0,0000412s |
| 100000 | 43800ns | 0,0438ms | 0,0000438s |
| 500000 | 1729100ns | 1,7291ms | 0,0017291s |
| 1000000 | 1734100ns | 1,7341ms | 0,0017341s |
| 10000000 | 1981500ns | 1,9815ms | 0,0019815s |



*Complejidad: La complejidad es O(n) ya que al estar implementado con una lista enlazada se necesita recorrer toda la lista para encontrar el elemento.*

Acceder a una ventana específica

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de ventanas | Tiempo en segundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 10400ns | 0,0104ms | 0,0000104s |
| 100000 | 12100ns | 0,0121ms | 0,0000121s |
| 500000 | 12400ns | 0,0124ms | 0,0000124s |
| 1000000 | 13400ns | 0,0134ms | 0,0000134s |
| 10000000 | 17600ns | 0,0176ms | 0,0000176s |

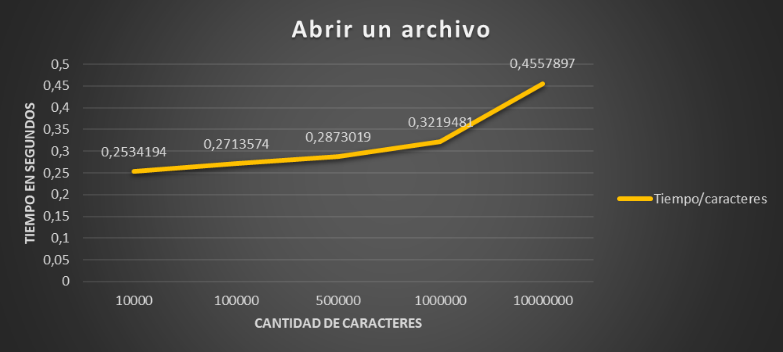


*Complejidad: La complejidad es O(n) ya que al estar implementado con una lista enlazada se necesita recorrer toda la lista para encontrar el elemento.*

***Abrir y guardar archivos***: Para abrir y guardar archivos se hicieron pruebas dependiendo de la cantidad de caracteres en el archivo. A continuación, se muestran las tablas de comparación del tiempo de ejecución respecto a la cantidad de caracteres y sus respectivas gráficas.

*Abrir un archivo*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de caracteres | Tiempo en nano segundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 253419400ns | 253,4194ms | 0,2534194s |
| 100000 | 271357400ns | 271,3574ms | 0,2713574s |
| 500000 | 287301900ns | 287,3019ms | 0,2873019s |
| 1000000 | 321948100ns | 321,9481ms | 0,3219481s |
| 10000000 | 455789700ns | 455,7897ms | 0,4557897s |



*Complejidad: La complejidad es O(n) ya que se tiene que escribir cada carácter uno por uno.*

*Guardar archivo*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de caracteres | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 212341800 | 212,3418 | 0,2123418 |
| 100000 | 206196300 | 206,1963 | 0,2061963 |
| 500000 | 218279400 | 218,2794 | 0,2182794 |
| 1000000 | 214049200 | 214,0492 | 0,2140492 |
| 10000000 | 227424900 | 227,4249 | 0,2274249 |

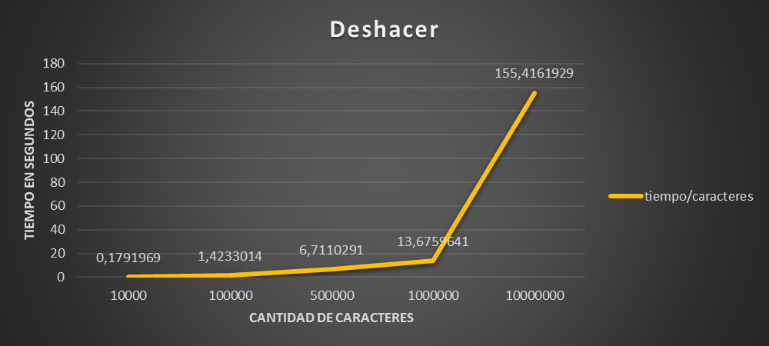


*Complejidad: La complejidad es O(n) ya que se tiene que guardar cada carácter código uno por uno.*

***Deshacer y rehacer:*** Para hacer las pruebas de los botones *deshacer* y *rehacer* se midió el tiempo de ejecución al momento de deshacer y rehacer cierta cantidad de caracteres. A continuación, se muestran las tablas de comparación del tiempo de ejecución respecto a la cantidad de caracteres y sus respectivas gráficas.

*Deshacer*

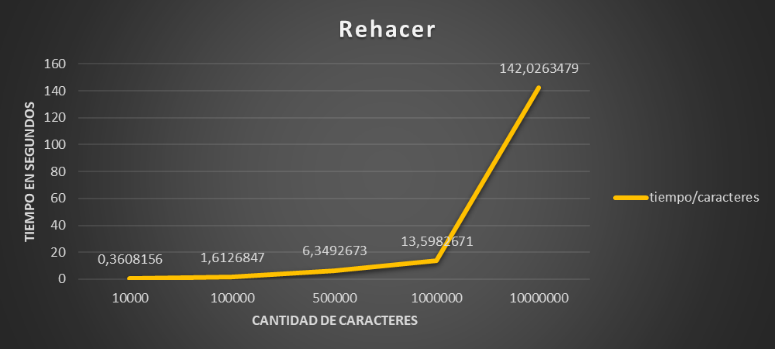
|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de caracteres | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 179196900ns | 179,1969ms | 0,1791969s |
| 100000 | 1423301400ns | 1423,3014ms | 1,4233014s |
| 500000 | 6711029100ns | 6711,0291ms | 6,7110291s |
| 1000000 | 1,3676E+10ns | 13675,9641ms | 13,6759641s |
| 10000000 | 1,5542E+11ns | 155416,193ms | 155,416193s |



*Complejidad: Complejidad: La complejidad es O(1) ya que está devolviendo y removiendo el último elemento ingresado en una pila.*

*Rehacer*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de caracteres | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 360815600ns | 360,8156ms | 0,3608156s |
| 100000 | 1612684700ns | 1612,6847ms | 1,6126847s |
| 500000 | 6349267300ns | 6349,2673ms | 6,3492673s |
| 1000000 | 1,3598E+10ns | 13598,2671ms | 13,5982671s |
| 10000000 | 1,4203E+11ns | 142026,348ms | 142,026348s |



*Complejidad: La complejidad es O(1) ya que está devolviendo y removiendo el último elemento ingresado en una pila.*

***Entradas del código:*** Las pruebas de las entradas del código se hicieron dependiendo de la cantidad de entradas. A continuación, se muestran las tablas de comparación del tiempo de ejecución respecto a la cantidad de entradas y sus respectivas gráficas.

*Ingresar entradas*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de entradas | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 2461600ns | 2,4616ms | 0,0024616s |
| 100000 | 6989900ns | 6,9899ms | 0,0069899s |
| 500000 | 17990200ns | 17,9902ms | 0,0179902s |
| 1000000 | 425298800ns | 425,2988ms | 0,4252988s |
| 10000000 | 1,1125E+10ns | 11125,1426ms | 11,1251426s |



*Complejidad: La complejidad es O(1) ya que cada entrada se está ingresando al final de una cola.*

*Devolver entradas y eliminarlas*

|  |  |  |  |
| --- | --- | --- | --- |
| Cantidad de entradas | Tiempo en nanosegundos | Tiempo en milisegundos | Tiempo en segundos |
| 10000 | 1232300ns | 1,2323ms | 0,0012323s |
| 100000 | 1672700ns | 1,6727ms | 0,0016727s |
| 500000 | 8944900ns | 8,9449ms | 0,0089449s |
| 1000000 | 8246100ns | 8,2461ms | 0,0082461s |
| 10000000 | 70042100ns | 70,0421ms | 0,0700421s |



*Complejidad: La complejidad es O(1) ya que se está devolviendo y removiendo el primer elemento de una cola.*

# roles y actividades

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Grupo | Rol | Actividad |
| Kevin J. Carranza | 5 | Experto | Desarrollo de la interfaz gráfica y los botones de deshacer y rehacer. |
| Ricardo A. Marino | 5 | Líder | Desarrollo del compilador de Brainfuck++. |
| David C. Cortés | 2 | Investigador | Coordinación del proyecto en Github y encargado de los problemas con Git. |

# DIFICULTADES Y LECCIONES APRENDIDAS

A pesar de las muchas dificultades que se presentaron en el camino, sin duda alguna la que tomó más tiempo resolver fue la de los problemas con la funcionalidad de deshacer y rehacer. Algunos de estos problemas se generaron debido a que en un principio no se hizo distinción entre cuando el usuario eliminaba parte del código con la tecla *Retroceso* y cuando el usuario eliminaba parte del código mediante la misma funcionalidad de *Retroceso*.

1. [↑](#footnote-ref-1)
2. https://github.com/ [↑](#footnote-ref-2)