Tecnológico de Costa Rica, Centro Académico Alajuela

Proyecto 1

Simpletron

Estudiantes:

Yurgen Isack Cambronero Mora

(2022128005)

Jorge Esteban Benavides Castro

(2022230697)

Profesores:

C. Martín Flores Gonzales

Allan Cascante

6 de octubre del 2022

Introducción

Este primer proyecto consiste en programar un sistema denominado como "Simpletron" que sistema solo puede ejecutar instrucciones que estén en el lenguaje máquina de Simpletron (LMS). Para el desarrollo de dicho algoritmo se va a disponer del lenguaje de programación Java, junto con el paradigma que propone la Programación Orientada a Objetos (POO). Con el objetivo de profundizar en el camino a seguir se tiene las secciones de Estrategia de la solución y Detalles de la implementación.

Entre los requerimientos necesarios para poder desarrollar exitosamente a Simpletron podemos encontrar a un acumulador en el cual se va a ir almacenando la información brindada por el usuario y que posteriormente va a utilizarse en las operaciones que se indiquen; además de esto la capacidad de leer correctamente los inputs del usuario, los cuales son una palabra de 5 dígitos acompañado de un signo (+ ó -) al inicio. Existen un total de 20 instrucciones distintas. La primera mitad de las palabras van a estar conformadas por el signo y dos dígitos que van a ser el indicador que señale que instrucciones ejecutar el comando existente dentro de Simpletron.

Para este proyecto la capacidad máxima de palabras, o comandos, permitidos es de 1000, por lo que cualquier lista de instrucciones que supere esta cantidad no podrá ejecutarse por completo. Respecto a esto, antes de que se pueda llegar a ejecutar cualquier de los comandos ingresado es importante que estos se cargan a la memoria de Simpletron. Una vez ahí ya se pueden ir cargando y ejecutando las instrucciones según se recorra esta última.

Considerando lo anterior se espera que poder crear un programa lo suficientemente eficiente para lograr ejecutar cualquier combinación de instrucciones de forma exitosa y en una pequeña cantidad de tiempo. Se espera también poder abarcar zonas más allá de las exhibidas en el enunciado del proyecto, es decir, mantener un control sobre posibles inputs que puedan llegar a vulnerar el sistema o generar una falla en el mismo, más sobre esto en la sección de Restricciones o suposiciones.

_ _

índice

_ _

Estrategia de la solución

Para poder desarrollar el código exitosamente lo primero que se hizo fue sentarnos a discutir el funcionamiento general que iba a tener el programa, es decir, las divisiones entre las clases que iban a existir y que papel iba a desempeñar cada una. Posterior a eso decidimos profundizar el funcionamiento del código a la hora de hacer el diagrama de las clases.

Cuando ya se contaba con un apoyo visuales empezó el proceso de programación, durante el cual se llegaron a presentar varios obstáculos y junto con ellos la necesidad de crear: nuevas variables, métodos y funcionamientos más específicos. A medida que avanza la programación se hacían pruebas rápidas para comprobar el correcto funcionamiento de las nuevas líneas.

Para terminar, se trabajó sobre el control de los inputs que podían llegar a romper código, es decir, se implemento la validación de las “palabras” hasta el final y una vez que el funcionamiento del código había sido corroborado.

Detalles de la implementación

1. Almacenamiento de instrucciones: Se recibían las instrucciones que luego serian compiladas basadas en el lenguaje LMS, el cual consistía en 6 dígitos, donde los primeros 3 indicaban la operación a realizar y los últimos 3 indicaban la dirección de memoria en la que se trabajaría.

2. Una vez recibidas todas las instrucciones, se iniciaba el proceso de compilación, donde al haber recibido la entrada -99999 se empezaría desde la línea o memoria 000 para ir así ejecutando el código, el cual se detendría cuando se encontrase la instrucción "+44".

3. Para esto se necesitó de 3 clases distintas, donde la primera seria la *Memory*, la cual se encargaría de almacenar datos e instrucciones, luego tendríamos la clase *Functions* la cual se encargaría de las distintas operaciones en su mayoría y finalmente tendríamos la clase *Simpletron*, la cual se encarga de recibir las líneas de instrucciones y luego compilar así el código, también, cabe decir que ciertas operaciones también fueron realizadas aquí por facilidad (como leer y escribir).

4. Para los saltos se realizó un cambio a la posición actual donde se encontraba la lista, ya que esta se recorre de manera local y se marcaron o realizaron varios trabajos con excepciones.

Restricciones o suposiciones

La primera suposición que podemos hacer sobre el código es que el mismo solo va a ejecutarse sobre la versión: Java 11. Dado esto, existe una restricción respecto al uso de distintas versiones ya que no se puede garantizar un correcto funcionamiento del sistema.

Otra restricción que se tuvo que incluir fue la invocación del método *close()* para el objeto *Scanner* puesto que al invocarlo no permitía utilizarlo más adelante a no ser que se incluyera una estructura *Try-Catch*. Por lo tanto, esto no llega a ser un problema ya que eventualmente se llamará a la interfaz de *Closeable* que cerrará el *Scanner.*

Otro factor por tomar en cuenta fue que supusimos que el *contador de ciclos* y la operación de*: bifurca hasta* podrían llegar a presentar un problema en su correcto

funcionamiento debido a que *contador de ciclos* es necesario para la operación 50 (*bifurca hasta*) pero este se invoca hasta después de la misma.

Finalmente, se restringieron los posibles inputs que puede ser ingresado por el usuario y se limitaron exclusivamente a aquellos que aparecen en el enunciado, es decir, si se ingresa una instrucción cuyos dos primeros dígitos después del signo no corresponden con comando válido se va a imprimir un mensaje de error.

Problemas encontrados

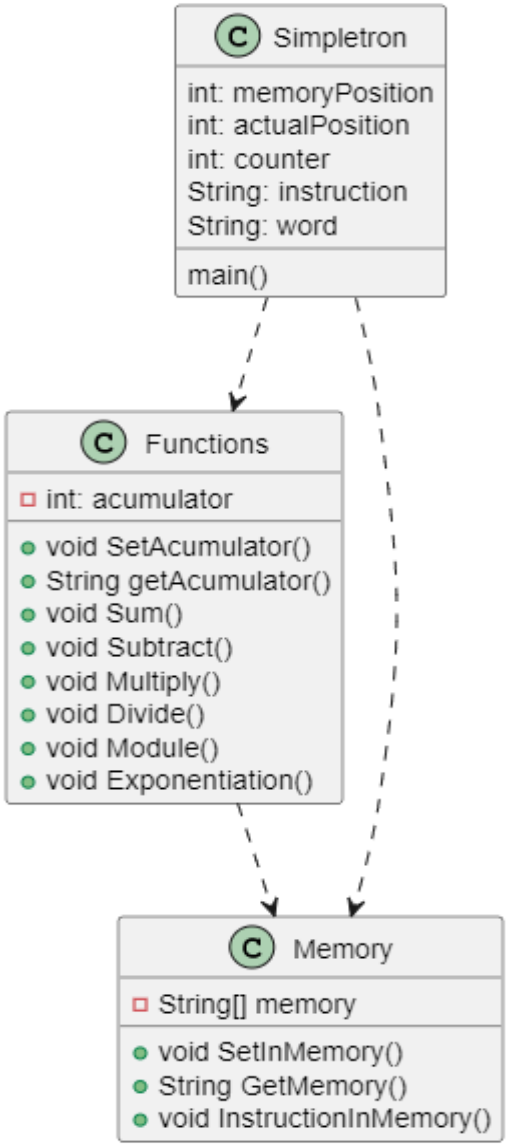
Durante la realización del código hubo varios obstáculos que dificultaron una correcta ejecución del algoritmo de Simpletron, algunos que pudimos solucionar fácilmente y otros que nos obligaron a repensar la manera en que estábamos abordando el problema.

Uno de estos problemas surgió durante la validación del comando que indica el fin de los *inputs* del usuario y el inicio del funcionamiento de Simpletron, puesto que a la hora de hacer la comparación del *input* con el valor: “-99999”, no lo identificada como correspondía. Para lograr resolver el problema en esa validación procedimos a quitarle el signo: “-” al *string* y convertir el restante en una variable de tipo *int*.

Otra problemática se presento a la hora de validar el tipo de instrucción dado que al recorrer el *switch* siempre entraba al caso *default* si el primer *input* era: “-99999”, se solucionó con una pequeña validación que revisa la memoria y si está vacía, de ser así el programa se cerraría con normalidad.

Las funciones matemáticas encargadas de las operaciones aritméticas (de la 30 hasta la 35), junto con la operación de carga, retornaban valores erróneos dado a que estaban trabajando con la posición de memoria en lugar de hacerlo con el valor contenido en esta última. Se resolvió, con facilidad, al indicarles a las funciones que trabajaran con el valor contenido en la memoria y no con la posición que ocupaban.

Diagramas de clases



Conclusiones y recomendaciones

A partir de lo elaborado durante la realización de este proyecto llegamos a la conclusión que se puede utilizar el paradigma de la programación orientada a objetos para realizar sistemas funcionales con relativa facilidad, como por ejemplo el objetivo de este proyecto, un lenguaje ensamblador básico.

Además de esto podríamos recomendar investigar sobre el lenguaje con el que se está trabajando debido a que podemos aprender sobre las posibles funciones incluidos en distintas librerías que facilitarían el proceso de programación como por ejemplo: *parseInt().*

Para concluir, siempre es recomendable mantener una consistencia a lo largo de todo el código para evitar el caos presente en el desarrollo de software. Una buena documentación y planeación del código puede facilitar increíblemente la carga de trabajo ante un proyecto.