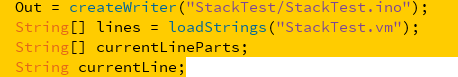
Documento De Funcionamiento Para Stack Test

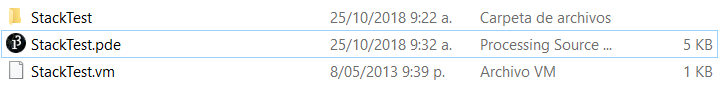
Este trabajo consiste en leer un archivo .VM y hacer una traducción por medio de processing para que el archivo pueda ser leído en la plataforma de Arduino.

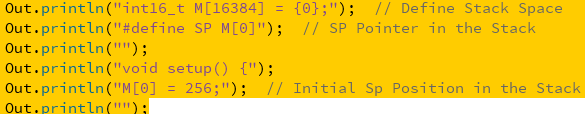
Para esto empezamos a utilizar algunas funciones específicas del lenguaje de processing( Java) para facilitar el proceso de lectura y traducción del archivo



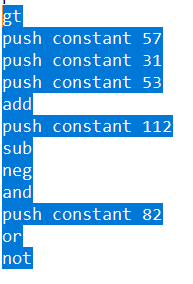
Primero creamos una variable PrintWriter que nos permitirá escribir dentro de un archivo de salida, Luego en el Setup del programa definimos el nombre del archivo de salida y la dirección, en este caso el archivo de salida llevará extensión. ino y se almacenará en dentro de la carpeta “Stacktest” con el nombre de StackTest.ino

Luego se crea un arreglo llamado lines que se encargara de leer cada una de las líneas de el archivo .vm que se encuentra dentro de la carpeta donde está ubicado el programa de modo tal que tendríamos algo así :

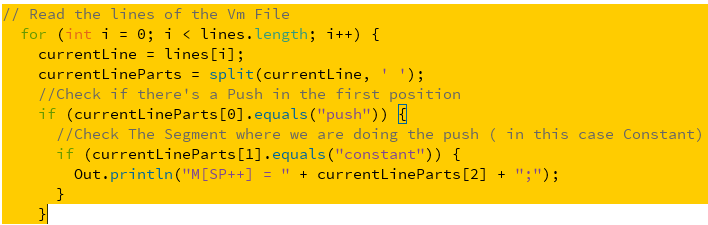




Antes de iniciar la traducción nos aseguramos de declarar las variables que necesitaremos en Arduino para el correcto funcionamiento del programa, en este caso necesitamos definir el tamaño del Stack, el Puntero de Sp en el stack, y la posición inicial de este puntero en el Stack.



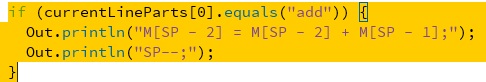
Para este ejemplo especifico debemos hacer varias operaciones logicas como lo son and,or,not y algunas operaciones aritméticas como en neg, seb , gt, eq, y lt.



Para esto tendremos el arreglo currentLineParts que nos servirá para almacenar por aparte cada parte de las diferentes líneas del archivo.Vm utilizando la función Split que se encargara de separar las cadenas cada vez que exista un espacio y la string, currentLine almacenará las diferentes posiciones de las lines del archivo.vm.

El código anterior se encarga de leer las diferentes líneas del archivo y si encuentra un push en la posición 0, y luego un constant en la posición 1 , creará una posición después del Sp en donde estará un carácter que se encuentra en la segunda posición del archivo, de modo que si tenemos un push constant 112, Añadiera un 112 despues del SP y asi sucesivamente con la cantidad de Push que el archivo.vm contenga.

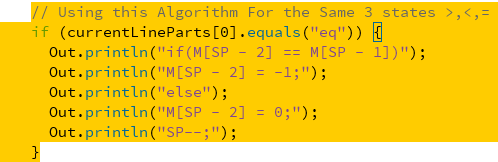
**Caso 1 – Add**

****

Se verificará si en el archivo hay un Add y en caso de que exista lo que haremos será sumar los valores que estaban en las 2 posiciones anteriores al stack , lo almacenaremos en la posición anterior al stack y finalmente reduciremos el stack.

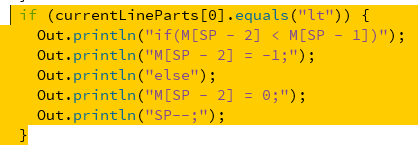
**Nota: Para los próximos 3 casos se utiliza el mismo algoritmo**

**Caso 2 – Eq**



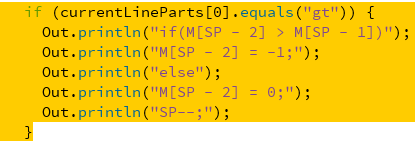
Se verificará si en el archivo hay un eq y en caso de que exista lo que haremos será comparar si la última posición ingresada en el Stack es igual a la penúltima y en caso de que sea cierto la última posición ingresada en el Stack tendrá un valor de -1 ( Que en la vm significa un True) de lo contrario entregara un 0 ( Correspondiente a 0 ). Finalmente reduciremos el stack .

**Caso 3 – Lt**

****

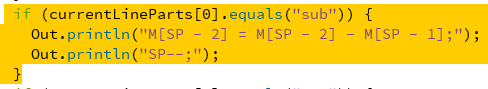
Se verificará si en el archivo hay un Lt y en caso de que exista lo que haremos será comparar si la última posición ingresada en el Stack es menor a la penúltima y en caso de que sea cierto la última posición ingresada en el Stack tendrá un valor de -1 ( Que en la vm significa un True) de lo contrario entregara un 0 ( Correspondiente a 0 ). Finalmente reduciremos el stack .

**Caso 4 – Gt**

****

Se verificará si en el archivo hay un Gt y en caso de que exista lo que haremos será comparar si la última posición ingresada en el Stack es mayor a la penúltima y en caso de que sea cierto la última posición ingresada en el Stack tendrá un valor de -1 ( Que en la vm significa un True) de lo contrario entregara un 0 ( Correspondiente a 0 ). Finalmente reduciremos el stack .

**Caso 5 – Sub**

****

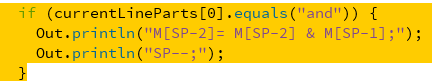
Se verificará si en el archivo hay un sub y en caso de que exista lo que haremos será restar los valores que estaban en las 2 posiciones anteriores al stack , lo almacenaremos en la posición anterior al stack y finalmente reduciremos el stack.

**Caso 6 – Neg**

****

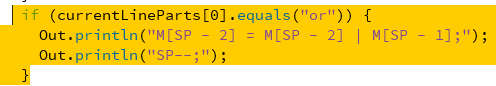
Si hay un Neg en el archivo invertiremos el signo del numero que esté justo después del Sp, Esta operación realmente saca el complemento A2 del numero que seria equivalente a invertir su signo.

**Caso 7 – And**

****

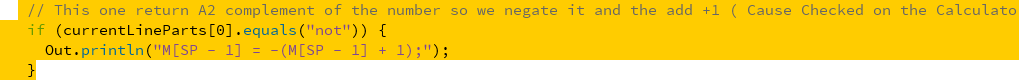
Se hará una operación lógica AND entre los últimos valores después del Sp y se decrementará el stack.

**Caso 8 – Or**

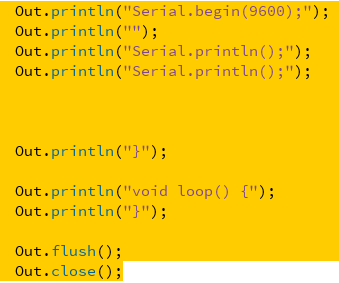
****

Se hará una operación lógica OR entre los últimos valores después del Sp y se decrementará el stack

**Caso 9 – Not**

****

Esta operación lo que hará será una compuerta lógica Not al último valor después del SP, para esto necesitara negarla es decir sacar su complemento A2 y luego sumarle 1 ( Esta conclusión se llego a travez de pruebas en la calculadora) .



Finalmente inicializaremos el serial de Arduino para poder visualizar los resultados del vector de prueba y compararlos con los de el archivo.vm , abriremos y cerraremos la función loop para un correcto funcionamiento dentro del IDE de Arduino, y utilizaremos la función Flush para asegurarnos que no quede ningún dato en el buffer y la función close para cerrar el archivo una vez todo el proceso halla finalizado,