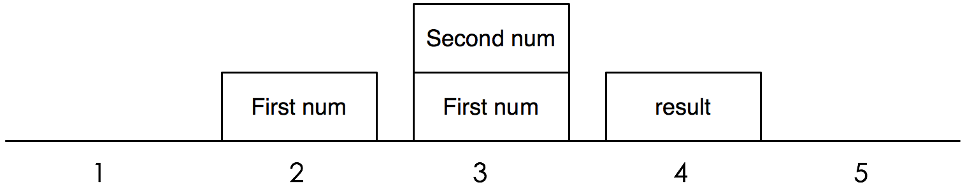
|  |  |
| --- | --- |
| **Instituto Tecnológico de Costa Rica**  **Ingeniería en Computación**  **Sede Regional San Carlos** | Compiladores e Intérpretes |
| Prof. Oscar Víquez Acuña. | Máquina Virtual simple para Monkey |

**Descripción:**

El proceso de compilación y ejecución de programas del lenguaje Monkey se divide en varias etapas. Inicialmente la compilación del lenguaje procesa detalles sintácticos y semánticos de algún código fuente de entrada y posteriormente genera el bytecode equivalente que será procesado por el intérprete para su ejecución.

La máquina virtual simple que se presenta a continuación, contempla algunos de los elementos básicos descritos en la teoría para organización en tiempo de corrida y generación de código. Al respecto, se contempla una serie de estructuras básicas necesarias para la ejecución de las instrucciones:

1. Módulo CODIGO: permite almacenar las instrucciones de bajo nivel del lenguaje que va a ejecutar la máquina virtual. Cada instrucción debe contemplar su nombre, y su parámetro en caso de que tenga. Debe contemplar métodos para alimentar la estructura de instrucciones a partir de un buffer de entrada.
2. Módulo PILA: permite almacenar valores de diversos tipos de datos simples en una pila, para eventualmente ser usados como parte de las operaciones de pila de la máquina virtual.



1. Módulo ALMACEN: permite almacenar tuplas de variables y valores (tipos simples) que serán utilizados por el intérprete para resolver las operaciones de búsqueda y almacenamiento de identificadores. Así mismo, almacenan referencias a métodos que deben llevar la "dirección" en el código que tiene dicho método. Los almacenes tienen la particularidad que pueden ser utilizados como diferentes instancias para diferentes contextos presentes en el código fuente de entrada de los programas que ejecuta.

La funcionalidad del intérprete está dictada por una serie de instrucciones que permiten la ejecución de comandos simples y usos de las estructuras mencionadas.

La siguiente es la lista de instrucciones que se permiten con su descripción, manejo de pila y algunas observaciones importantes que deben considerarse al momento de escribirlas:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INSTRUCCIÓN | PARÁMETROS | DESCRIPCIÓN | DEBE HABER EN LA PILA | QUÉ DEJA EN LA PILA | Observaciones |
| PUSH\_LOCAL\_I | *name* | Declara una variable local de tipo entero en el almacén | [] | [] |  |
| PUSH\_LOCAL\_C | *name* | Declara una variable local de tipo char en el almacén | [] | [] |  |
| PUSH\_GLOBAL\_I | *name* | Declara una variable global de tipo entero en el almacén | [] | [] |  |
| PUSH\_GLOBAL\_C | *name* | Declara una variable global de tipo char en el almacén | [] | [] |  |
| DEF | *name* | Declara un método en el almacén | [] | [] | Debe considerar guardar la dirección de la definición del método para efectos de saltos posteriores al hacer llamadas. |
| LOAD\_CONST | *const* | Coloca el valor de la constante en el tope de la pila | [] | [const] | Cualquier constante de cualquier tipo de dato permitido |
| LOAD\_FAST | *varname* | Coloca el valor del contenido de la variable en la pila | [] | [const] |  |
| STORE\_FAST | *varname* | Escribe el contenido del tope de la pila en la variable local | [const] | [] |  |
| STORE\_GLOBAL | *varname* | Escribe el contenido del tope de la pila en la variable global | [const] | [] |  |
| LOAD\_GLOBAL | *name* | Carga en el tope de la pila o el valor de la variable o la referencia a la función | [] | [const | ref] | Nótese que el parámetro no dice varname, sino name porque globales serán tanto las variables como las funciones. |
| CALL\_FUNCTION | *numparams* | Realiza un salto a la dirección de código de la función | […params…, funcref] | [val]  si es función y retorna | Nótese que ..params… son varias constantes que se pueden encontrar en la pila a partir del tope seguidas por la referencia a la función  DEBE CONSIDERARSE LA DIRECCIÓN DE RETORNO |
| RETURN\_VALUE |  | Toma el valor en el tope de la pila, retorna (salto) a la dirección en el código posterior a la ejecución de la función y coloca el valor sacado del tope y lo vuelve a insertar | [const] | [const] | Normalmente un RETURN\_VALUE va precedido de un LOAD con el valor a retornar. El return debe sacar todo aquello de la función que haya quedado en la pila referente a esa función y debe conocer la DIRECCIÓN DE RETORNO para poder saltar |
| END |  | Termina la ejecución del programa |  |  |  |
| COMPARE\_OP | *op* | Realiza una comparación booleana según el op que reciba | [oper2, oper1] | [result] | El resultado es algo que represente o un TRUE o un FALSE |
| BINARY\_SUBSTRACT |  | Realiza una resta de operandos | [oper2, [oper1] | [result] |  |
| BINARY\_ADD |  | Realiza una suma de operandos | [oper2, oper1] | [result] |  |
| BINARY\_MULTIPLY |  | Realiza una multiplicación de operandos | [oper2, oper1] | [result] |  |
| BINARY\_DIVIDE |  | Realiza una división entera de operandos | [oper2, oper1] | [result] |  |
| BINARY\_AND |  | Realiza un AND lógico | [oper2, oper1] | [result] |  |
| BINARY\_OR |  | Realiza un OR lógico | [oper2, oper1] | [result] |  |
| BINARY\_MODULO |  | Realiza el cálculo del cociente de la división de dos operandos | [oper2, oper1] | [result] |  |
| JUMP\_ABSOLUTE | *target* | Salta a la línea de código indicada por “target” | [] | [] | “target” es un índice indicado en el archivo que debe coincidir con el # de instrucción una vez almacenado el código en memoria |
| JUMP\_IF\_TRUE | *target* | Si el tope de la pila es True, salta a “target” | [valueTF] | [] | “target” es un índice indicado en el archivo que debe coincidir con el # de instrucción una vez almacenado el código en memoria |
| JUMP\_IF\_FALSE | *target* | Si el tope de la pila es False, salta a “target” | [valueTF] | [] | “target” es un índice indicado en el archivo que debe coincidir con el # de instrucción una vez almacenado el código en memoria |
|  |  |  |  |  |  |

*Las instrucciones descritas anteriormente representan solo un fragmento de las capacidades del lenguaje Monkey y deberán ser ampliadas para aumentar la funcionalidad del intérprete, según se estipule en el documento de generación de código del proyecto.*

La máquina virtual simple para Monkey leerá y correrá instrucciones a partir de un archivo de texto que contiene una secuencia de las mismas con las condiciones y requerimientos previamente descritos. Este archivo de texto es el producto de la etapa de generación de código.

El archivo tendrá un formato específico donde habrá:

1. Etiquetas que indiquen el inicio de los métodos usando el nombre del mismo y dos puntos
2. Cada instrucción tendrá o no solo un parámetro separado por un espacio en blanco
3. Le instrucción vendrá o no en mayúscula