**Instituto Tecnológico de Costa Rica**

I tAREA pROGRAMADA (pROGRAMACIÓN Imperativa)

Sede San Carlos 18/04/2017

## Unidad de Computación

# **Jose Antonio Herrera Ramírez**

# Introducción

Hoy en día la mayoría de los proyectos software se desarrollan dentro del paradigma imperativo. Este paradigma se caracteriza por ser dinámico, ya que el programa puede pasar por distintos estados durante la ejecución del mismo y se puede alterar fácilmente su comportamiento, o bien los espacios de memoria que este procesa.

En este proyecto se realizará un programa para procesar instrucciones bytecode definidas en el alcance del proyecto, las cuales serán generadas en un archivo externo por el compilador de Python.

El lenguaje bajo el cual se desarrollará el proyecto es ‘C’, el cual permite un manejo complejo de los espacios de memoria o de los estados del programa.

# Análisis del problema

El programa a desarrollar debe ser capaz de leer los archivos bytecode externos, interpretar las instrucciones y realizar todas las operaciones que sean necesarias para la ejecución de estas.

Se debe considerar un manejo muy preciso de la memoria, ya que en el lenguaje se pueden dar muchos errores por una mala administración de esta.

El programa debe utilizar estructuras de datos donde se puedan guardar las instrucciones del archivo en memoria a fin de una mejor manipulación de los datos, de manera que una vez empezado el programa las instrucciones se persistan en memoria para su posterior procesamiento.

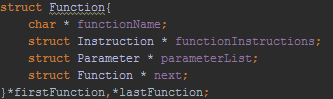
También se deberá poder extraer los datos del archivo de una manera limpia y consistente, distinguiendo los datos de diferentes tipos como float, int, string, char ,etc. Asimismo, se deben poder evitar los comentarios en el archivo y demás inconsistencias que se puedan presentar en este.

# Solución del problema

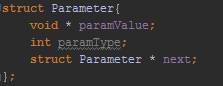
El proceso de lectura de los archivos se llevó a cabo mediante métodos de la librería estándar para el manejo de archivos, este proceso consiste en una lectura de línea por línea del archivo y la descomposición de la línea para su almacenamiento adecuado en memoria.

Para la persistencia de las instrucciones se utilizaron distintas estructuras de datos que permitirán almacenar todo en memoria, entre estas están:

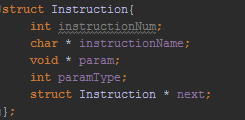
**Estructura para las Funciones o etiquetas del archivo**



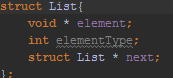
**Estructura para los parámetros de cada función**



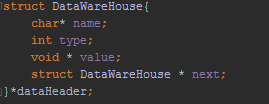
**Estructura para las instrucciones pertenecientes a cada función**

****

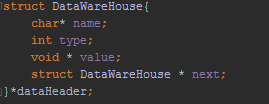
**Estructura para el soporte de listas en el programa**

****

**Estructura que simulara el almacén de datos para guardar variables del programa, etc.**

****

**Estructura pila para el procesamiento de instrucciones**



Estas estructuras permitirán el acceso a los datos guardados en memoria de una manera eficiente.

El proceso de ejecución del programa debería ser el siguiente:

1. Leer el archivo y extraer los datos para su almacenado en memoria.
2. Leer los datos en memoria y ejecutarlos según las instrucciones de cada función, comenzando con la función Main().

# Análisis de los resultados

El alcance del proyecto no se logró terminar al 100%, por lo que existen distintas funciones que no se implementaron en el programa.

Además no existe la función principal que leerá los datos en memoria y los ejecutara uno por uno, por lo que es imposible ver la ejecución del programa.

La siguiente tabla mostrará el conjunto de funciones que se lograron desarrollar resaltadas en verde y aquellas que no se implementaron en el programa sin resaltar.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INSTRUCCIÓN | PARÁMETROS | DESCRIPCIÓN | DEBE HABER EN LA PILA | QUÉ DEJA EN LA PILA |
| LOAD\_CONST | *const* | Coloca el valor de la constante en el tope de la pila | [] | [const] |
| LOAD\_FAST | *varname* | Coloca el valor del contenido de la variable en la pila |  |  |
| STORE\_FAST | *varname* | Escribe el contenido del tope de la pila en la variable | [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] |
| CALL\_FUNCTION | *numparams* | Realiza un salto a la dirección de código de la función | […params…, funcref] |  |
| 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] |
| COMPARE\_OP | *op* | Realiza una comparación booleana según el op que reciba | [oper2, oper1] | [result] |
| 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] |
| STORE\_SUBSCR |  | Realiza la operación:  array[index] = value | [index, array, value] | [] |
| BINARY\_SUBSCR |  | Carga en el tope de la pila el elemento de un arreglo en el índice indicado | [index,array] | [array[index]] |
| JUMP\_ABSOLUTE | *target* | Salta a la línea de código indicada por “target” | [] | [] |
| JUMP\_IF\_TRUE | *target* | Si el tope de la pila es True, salta a “target” | [valueTF] | [] |
| JUMP\_IF\_FALSE | *target* | Si el tope de la pila es False, salta a “target” | [valueTF] | [] |
| BUILD\_LIST | *elements* | Construye una lista con “elements” cantidad de elementos | [elem1.. elemN] | [lista] |

# Conclusiones y Recomendaciones

El paradigma imperativo se caracteriza por su flexibilidad de estados del programa y la implementación de cambios a estos estados mediante sentencias.

El lenguaje utilizado es un exponente muy robusto y es un buen ejemplo de un lenguaje robusto dentro del paradigma imperativo.

# Bibliografía

Concepcion, U. (s.f.). *Manejo de Archivos en C*. Obtenido de Informatica.utem.cl: http://informatica.utem.cl/~mcast/PROGRAMACION/PROGRAV/2007/ARCHIVOS/archivos-1.pdf

Coruña, U. (s.f.). *El lenguaje de programación C.* La Coruña.

.