CANVAS COMPILER

DOCUMENTACIÓN

Diseño de Compiladores
Profesora Elda Quiroga / Hector Ceballos

Enrique Marroquín Artezán
A01191578

Roel Castaño Moreno
A01191163

3 de Mayo del 2017

Indice

Descripción del Proyecto

Descripción del Proyecto

Misión y Visión

La misión de Canvas es la de ser un lenguaje de programación con una curva de aprendizaje baja. Que al mismo tiempo ofrezca características llamativas para audiencias artísticas. Logramos esto al mantener una sintaxis clara y descriptiva.

La visión es de integrar a individuos con habilidades visuales a la comunidad de la programación. Asimismo, facilitar el proceso de aprendizaje de conceptos de la programación con el apoyo de representaciones gráficas.

Objetivo

El objetivo principal de este lenguaje es incentivar el uso de la programación a personas sin experiencia previa por medio del uso de representaciones visuales de los resultados de sus programas. Al tener una salida visual, consideramos que el lenguaje cae dentro del área de lenguajes gráficos.

Alcance del Proyecto

Canvas contiene todos los elementos básicos de un lenguaje de programación, adicionalmente incluye el elementos propios del lenguaje.

- Expresiones aritméticas, lógicas y relacionales.
- Estatutos de impresión
- Estatutos de control de flujo (ciclos, decisiones)
- Elementos de cambio de contexto (funciones, métodos parametrizados)
- Manejo de elementos no atómicos (arreglos)
- Elementos propios del tipo de proyecto (output gráfico)
- Documentación

Análisis de Requerimientos y Casos de Uso Generales

Requerimientos funcionales

- La compilación de un programa Canvas genera un archivo de salida csv
- Es posible desplegar programas en ventanas gráficas
- Es posible utilizar una terminal para compilar el programa a compilar
- La máquina virtual interpreta el código intermedio y lo ejecuta

Requerimientos funcionales

- Los programas se escriben en archivos de texto con extensión .cv
- La sintaxis debe ser fácil de entender para un principiante de la programación
- El proyecto Canvas permanecerá open source

Descripción de los Principales Casos de Uso

Compilador:

- Utiliza al escáner para reconocer y filtrar tokens pertenecientes al lenguaje propio
- Y rechaza el uso de tokens no especificadas
- Utiliza al parser para interpretar las reglas de gramática que se presentan dentro del programa
- Interpreta la semántica del lenguaje y verifica que esta sea utilizada correctamente
- Genera un código intermedio en forma de cuadruplos y tablas de memoria

Máquina Virtual:

- Toma el archivo de salida generado por el compilador y lo lee
- Prepara la memoria virtual utilizando la información otorgada por el compilador y plasmada en el archivo
- Interpreta los cuádruplos generados por el compilador y toma acciones dependiendo del tipo de cuádruplo leído
- Crea contextos de ejecución para la utilización de funciones. Guardando sus memorias correspondientes

Usuario:

- Escribe programas utilizando las reglas de gramática definidas por Canvas
- Compila sus archivos donde escribió el programa utilizando una línea de comandos
- Ejecuta el archivo generado por el compilador utilizando la máquina virtual de Canvas

Descripción del Proceso de Desarrollo

Para el desarrollo del proyecto, se llevó a cabo un desarrollo de tipo incremental, utilizando el popular control de versiones Git y la plataforma de Github. Generalmente cuando se trabajaba en diferentes carecteristicas, utilizabamos branches y pull requests para hacer la unión del trabajo.

Se llenaban bitácoras semanales para ir documentando el progreso del proyecto.

Bitácora

```
___FEB 26 - MAR 4___
+ Primer implementacion scanner y parser

___MAR 5 - MAR 11___
~ MODIFICACION DE GRAMATICA: en 'expression' para soportar terminos constantes (ver: 'factor_value' en
la gramatica)
~ MODIFICACION DE GRAMATICA: en 'expression' para soportar solo un 'exp'
~ MODIFICACION DE GRAMATICA: en 'shape', al especificar center, deberia ser un id de un point, en lugar
de definir un point
~ MODIFICACION DE GRAMATICA: en 'shape', al especificar color, deberia ser un id de un color, en lugar
de definir un color
~ MODIFICACION DE LEXICO: cambiar a 'yesno' de posicion, para que este arriba de 'yes' y 'no'
~ MODIFICACION DE LEXICO: cambiar a 'double' de posicion, para que este arriba de 'int'
~ MODIFICACION DE GRAMATICA: en 'declaration', para soportar declaracion de 'color'
~ MODIFICACION DE GRAMATICA: en 'los *_assignment', que pidiera un id antes de hacer un assignment corto
~ MODIFICACION DE GRAMATICA: en 'for-loop' para que soporte listas
~ MODIFICACION DE GRAMATICA: en 'var_assignment' para que soporte asignacion de yes/no
~ MODIFICACION DE GRAMATICA: en 'exp' quitar el null
+ Agregar nuevos tests
+ Agregar feature para leer tests de archivos separados
+ Implementacion de tabla de variables



___MAR 12 - MAR 18___
+ Implementacion de clases Var, Function y FuncitonDirectory
+ Divided globals into vars and functions
```

```
___MAR 19 - MAR 25___
+ Creacion de tabla de variables, funciones y directorio de funciones dentro del parser
~ Modificacion de gramatica en p_function para aceptar parametros y vars en el scope de la funcion


___MAR 26 - ABR 1___
+ Creacion de clase cuadruplos y reading_controller
+ Implementacion de Cubo Semantico
~ Reducimos conflictos de 4 -> 3 shift/reduce y 76 -> 19 reduce/reduce


___ABR 2 - ABR 8____
+ Integra if, elsif, y else a quadruplos
+ Integra while quads

___ABR 9 - ABR 15___
___ABR 16 - ABR 20___
+ Cuadruplos de Arreglos y MV
~ Reducimos condflictos de 3 -> 1 shift/reduce y 19 -> 0 reduce/reduce

___ABR 24 - ABR 28___
+ Arreglos y MV
+ IR

___MAY 1 - MAY 3___
+ Arreglos y MV
+ Aplicación output gráfico
```

Aprendizajes

“Trabajando en un proyecto con esta magnitud me di cuenta de la importancia de una buena metodología de desarrollo, con la cual se asegure calidad de código. Asimismo aplique conocimientos de mis previas clases de carrera. Disfrute el hecho de poder pensar en ideas abstractas y plasmarlas en código, conectando las diferentes partes unas con otras” - Enrique

“Aprendí mucho sobre la importancia de la disciplina, hacer todo a su tiempo. Un proyecto de esta magnitud requiere de una cantidad de trabajo constante por un periodo largo de tiempo.

No se puede llevar a cabo solamente en una o dos semanas, requiere de más tiempo. Es importante la disciplina y administración de tiempos para las distintas partes del proyecto.” -Roel

Descripción del Lenguaje

Nombre del Lenguaje

Descripción genérica de las principales características del lenguaje

Canvas es una palabra para describir el lienzo cuyo artista utiliza para expresarse de manera creativa. Este compilador intenta emular esta experiencia en un entorno de programación. Canvas cuenta con los componentes básicos de cualquier lenguaje de programación, pero su factor diferenciador se encuentra en su salida. Los usuarios del lenguaje podrán ejecutar programas y ver resultados de manera visual en una ventana gráfica.

Descripción de los errores - Compilación

NameError - Variable ya definida
NameError - Función ya definida
TypeError - Operación no válida
SyntaxError - Token no reconocido

Descripción de los errores - Ejecución

TypeError - Operación no válida

IndexOutOfBounds - Índice de arreglo está fuera de los límites

Descripción del Compilador

Equipo de Cómputo, Lenguaje y Utilerias.

Marca: Apple.
Modelo: MacBook Pro.
Sistema Operativo: OSX.
Lenguaje Utilizado: Python 2.7.
Análisis Léxico y Semántico: PLY.
Libreria de Salida Gráfica: graphics.py

Descripción del Análisis de Léxico

```
reserved_words = {
    'int'       : 'INT',
    'dec'       : 'DEC',
    'string'    : 'STRING',
    'point'     : 'POINT',
    'circle'    : 'CIRCLE',
    'rectangle' : 'RECTANGLE',
    'canvas'    : 'CANVAS',
    'program'   : 'PROGRAM',
    'function'  : 'FUNCTION',
    'paint'     : 'PAINT',
    'print'     : 'PRINT',
    'for'       : 'FOR',
    'each'      : 'EACH',
    'in'        : 'IN',
    'if'        : 'IF',
    'elsif'     : 'ELSIF',
    'else'      : 'ELSE',

    'while'     : 'WHILE',
    'finish'    : 'FINISH',
    'main'      : 'MAIN',
    'while'     : 'WHILE',
    'return'    : 'RETURN',
    'returns'   : 'RETURNS',
}

tokens = [
    'EQUALS',
    'L_BRACKET',
    'R_BRACKET',
    'G_THAN',
    'L_THAN',
    'NOT_EQUALS',
    'EQUALS_EQUALS',
    'G_THAN_EQUALS',
    'L_THAN_EQUALS',
```

```
def t_NOT_EQUALS(t):
    r'!='
    return t

def t_G_THAN_EQUALS(t):
    r'>='
    return t

def t_L_THAN_EQUALS(t):
    r'<='
    return t

def t_G_THAN(t):
    r'>'
    return t

def t_L_THAN(t):
    r'<'
    return t

def t_PLUS(t):
    r'\+'
    return t

def t_MINUS(t):
    r'\-'
    return t

def t_DIV(t):
    r'/'
    return t

def t_MULT(t):
    r'\*'
    return t
```

```
    'PLUS',
    'MINUS',
    'DIV',
    'MULT',
    'L_PAR',
    'R_PAR',
    'COMMA',
    'INT_VAL',
    'DEC_VAL',
    'STRING_VAL',
    'YESNO',
    'YESNO_VAL',
    'VAR_IDENTIFIER'] +
list(reserved_words.values())
t_ignore  = ' \t'

def t_EQUALS_EQUALS(t):
    r'=='
    return t

def t_EQUALS(t):
    r'\='
    return t

def t_L_PAR(t):
    r'\('
    return t

def t_R_PAR(t):
    r'\)'
    return t

def t_L_BRACKET(t):
    r'\['
    return t

def t_R_BRACKET(t):
    r'\]'
    return t
```

```
def t_COMMA(t):
    r','
    return t

def t_POINT(t):
    r'\.'
    return t

# REGULAR EXPRESSIONS
def t_DEC_VAL(t):
    r'[0-9]+[\.][0-9]+'
    return t

def t_INT_VAL(t):
    r'[0-9]+'
    return t

def t_STRING_VAL(t):
    r'\"([^\\\n]|(\\.))*?\"'
    return t

def t_YESNO(t):
    r'yesno'
    return t

def t_YESNO_VAL(t):
    r'(no|yes)'
    return t

def t_VAR_IDENTIFIER (t):
    r'[a-zA-Z]+[a-zA-Z0-9]*(_[a-zA-Z0-9]*)*'
    t.type = reserved_words.get(t.value,
'VAR_IDENTIFIER')
    return t

# Define a rule so we can track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
```

Descripción del Análisis de Sintaxis.

Gramática formal

```
PROGRAM ::= 'program' 'id' GLOBALS FUNCTIONS 'main' BLOCK_WITH_DECLARATION 'finish'
```

```
GLOBALS ::= DECLARATION*
```

FUNCTIONS ::= FUNCTIONS*
FUNCTION ::= 'function' 'id' '(' (TYPE 'id' (',' TYPE 'id')*)? ')' 'returns' TYPE BLOCK_WITH_DECLARATION
VAR ::= TYPE 'id' LIST_INDEX '=' EXPRESSION
SHAPE ::= ('circle' | 'rectangle') 'id' '(' EXPRESSION ',' EXPRESSION ')'
BLOCK_WITH_DECLARATION ::= '[' DECLARATION+ STATEMENT* ']'
BLOCK ::= '[' STATEMENT* ']'
STATEMENT ::= ASSIGNMENT | CONDITIONAL | PRINT | LOOP | SHAPE | PAINT | RETURN
ASSIGNMENT ::= VAR_ASSIGNMENT | POINT_ASSIGNMENT | CANVAS_ASSIGNMENT
VAR_ASSIGNMENT ::= 'id' ('[' [0-9]+ ']')? '=' (EXPRESSION | 'id')
DECLARATION ::= VAR | SHAPE | CANVAS
CANVAS ::= 'canvas' 'id'
EXPRESSION ::= EXP (('>' | '<' | '!=' | '==' | '>=' | '<=') EXP)?
EXP ::= (TERM (('+' | '-') TERM)*)?
TERM ::= FACTOR (('*' | '/') FACTOR)*
TYPE ::= 'int' | 'dec' | 'string' | 'yesno'
FACTOR ::= '(' EXPRESSION ')' | ('+' | '-')* (FACTOR_VAR | 'int_val' | 'dec_val' | 'yesno_val' | 'string_val')
FACTOR_VAR ::= 'id' LIST_INDEX_EXP FUNCTION_CALL
LIST_INDEX ::= ('[' [0-9]+ ']')?
LIST_INDEX_EXP ::= ('[' EXPRESSION ']')?
FUNCTION_CALL ::= ('(' 'id' (',' 'id')* ')')?
CONDITIONAL ::= 'if' '(' EXPRESSION ')' BLOCK ('elsif' '(' EXPRESSION ')' BLOCK)* ('else' BLOCK)?
PRINT ::= 'print' '(' (EXPRESSION | 'id') (',' (EXPRESSION | 'id'))* ')'
PAINT ::= 'paint' 'id'
RETURN ::= 'return' EXPRESSION
FOR_LOOP ::= 'for' 'each' 'id' 'in' 'id' BLOCK
WHILE_LOOP ::= 'while' '(' EXPRESSION ')' BLOCK

Descripción de Generación de Código Intermedio y Análisis Semántico.

Codigo de Operacion y Direcciones Virtuales Asociadas a los elementos de Código

Asociar operando a direccion virtual

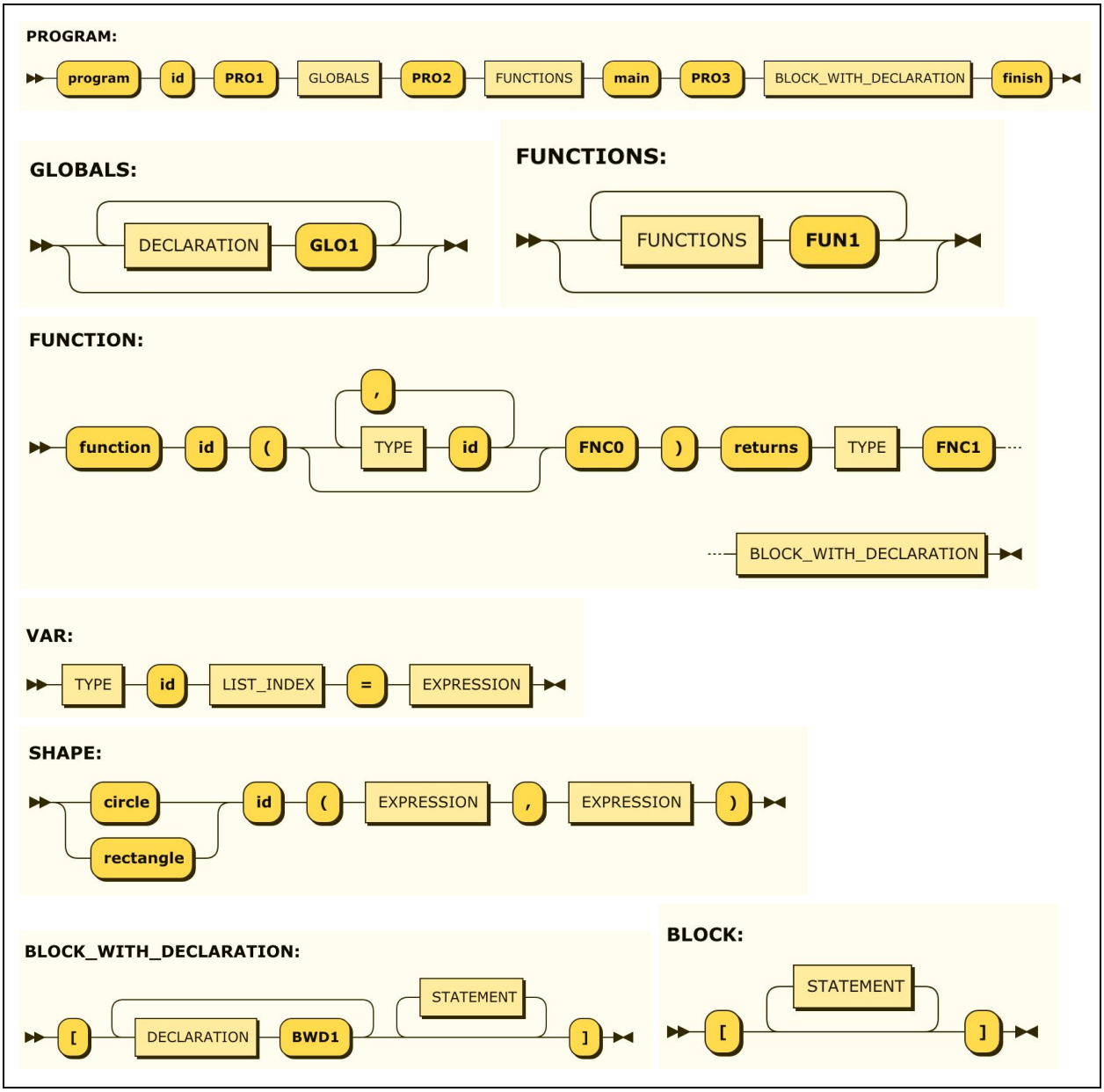
```
def p_push_operand(p):
    '''
    push_operand :
    '''
    var_name = p[-2]
    var_type = p[-3]
    if var_name in function_dir.global_function.variables:
        raise NameError("Name already defined", var_name)
    else:
        if p[-1]: #array definition
            virt_address = memory_controller.generate_arr_address(ALLOC_SCOPE, var_type,
p[-1])
        else:
            virt_address = memory_controller.generate_var_address(ALLOC_SCOPE, var_type)
        quad_controller.read_operand(virt_address)
        quad_controller.read_type(var_type)
        p[0] = virt_address
```

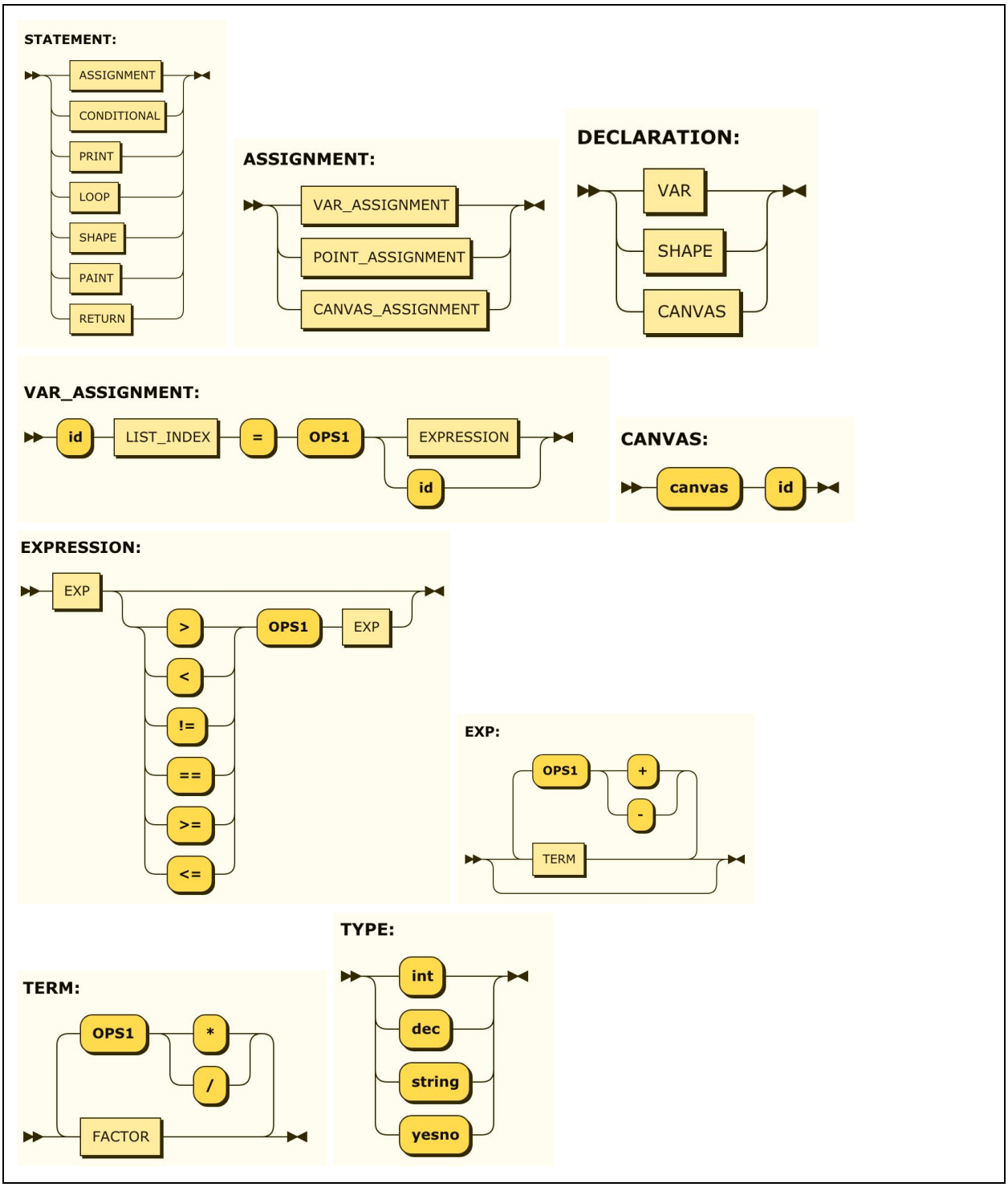
Memory Controller, method to generate virtual address:

```
class MemoryController:

    temp_memory = MemoryMap()
    local_memory = MemoryMap()
    global_memory = MemoryMap()
    const_memory = ConstantsTable()

    '''
    string which maps our virtual addresses:

    ADDRESS = MEM_SEGMENT + VAR_TYPE_CODE + SUBINDEX
    e.g. "gin23" is global int at index 23
    '''
    def generate_var_address(self, var_segment, var_type):
        next_avail = 0
        if var_segment == TEMP_SEGMENT:
            next_avail = self.temp_memory.types[var_type]
            self.temp_memory.types[var_type] += 1
        elif var_segment == GLOBAL_SEGMENT:
            next_avail = self.global_memory.types[var_type]
            self.global_memory.types[var_type] += 1
        elif var_segment == LOCAL_SEGMENT:
            next_avail = self.local_memory.types[var_type]
            self.local_memory.types[var_type] += 1
```

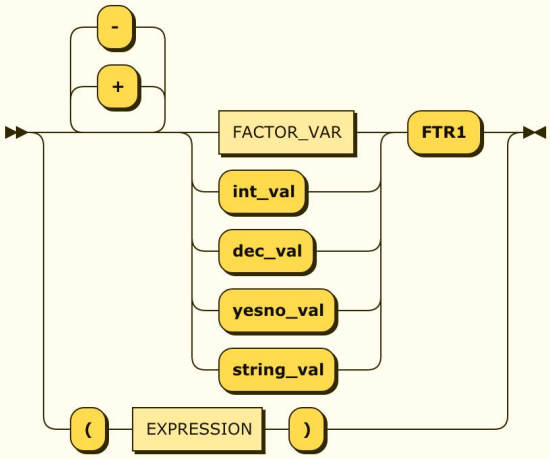
```
        return var_segment + var_type[0:2] + str(next_avail)
```

Diagramas de Sintaxis



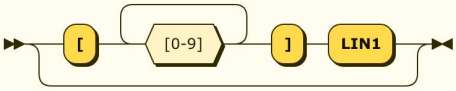

STATEMENT:
ASSIGNMENT
CONDITIONAL
PRINT
LOOP
SHAPE
PAINT
RETURN
ASSIGNMENT:
VAR_ASSIGNMENT
POINT_ASSIGNMENT
CANVAS_ASSIGNMENT
DECLARATION:
VAR
SHAPE
CANVAS
VAR_ASSIGNMENT:
id
LIST_INDEX
=
OPS1
EXPRESSION
id
CANVAS:
canvas
id
EXPRESSION:
EXP
>
<
!=
==
>=
<=
OPS1
EXP
EXP:
OPS1
+
-
TERM
TERM:
OPS1
*
/
FACTOR
TYPE:
int
dec
string
yesno

FACTOR:



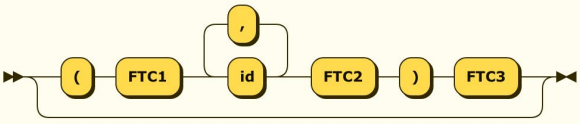
FACTOR_VAR:

LIST_INDEX:

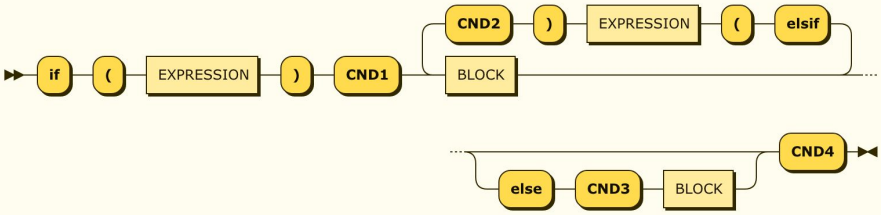


LIST_INDEX_EXP:

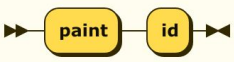
FUNCTION_CALL:



CONDITIONAL:

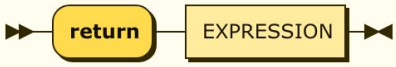


PAINT:

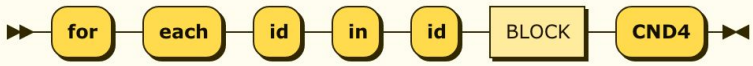


PRINT:

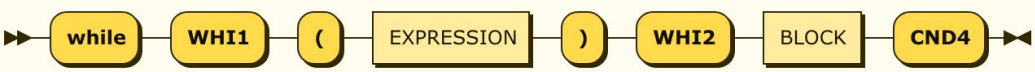
RETURN:



FOR_LOOP:



WHILE_LOOP:



```
PRO1 - push 'main' quad, save index in jump stack
PRO2 - assign mem_maps to temp_function, clear mem_maps and clear temp_function
PRO3 - pop from jump_stack, fill(main_quad_index, current_counter)
GLO1 - push DECLARATION to temp_function_table
FUN1 - assign mem_maps to temp_function, clear mem_maps and clear temp_function
FNC0 - save declaration of temp_var in func_table
FNC1 - save declaration of temp_func in func_dir
BWD1 - save declaration of temp_var in func_table
LIE1 - push fake_bottom to operator stack
LIE2 - pop fake_bottom from operator stack, gen_quad('VER', index, arr_size),
gen_quad('ADDBASE', index, base, temp)
LIN1 - alloc index size to tmp_var
FTC1 - push fake_bottom to operator stack
FTC2 - pop fake_bottom from operator stack
FTR1 - operand_stack.push(factor), type_stack.push(factor)
FCT3 - gen_quad('GOSUB', virt_addr, func_index_in_quads)
CND1 - gen_quad('GOTOF', res)
CND2 - 1,2 = jump_stack.pop(), jump_stack.pop(); fill_quad(1, counter), jump_stack.push(2)
CND3 - gen_quad('GOTO'), fill(false, counter), jump_stack(counter - 1)
CND4 - e = jump_stack.pop(), fill(e, counter)
WHI1 - jump_stack.push(counter)
WHI2 - 1,2 = jump_stack.pop(), jump_stack.pop(); fill_quad("GOTO", 2), fill(2< counter)
OPS1 - operand_stack.push(op)
```

Cubo Semantico

```
SemanticCube = [[[-1 for j in range(len(operator_dict))] for k in range(len(type_dict))] for
m in range(len(type_dict))]

math_opers = ['+', '-', '*', '/']

for op in math_opers:
    SemanticCube[type_dict['int']][type_dict['int']][operator_dict[op]] = 'int'
    SemanticCube[type_dict['dec']][type_dict['dec']][operator_dict[op]] = 'dec'
    SemanticCube[type_dict['dec']][type_dict['int']][operator_dict[op]] = 'dec'
    SemanticCube[type_dict['int']][type_dict['dec']][operator_dict[op]] = 'dec'

comp_opers = ['<', '>', '<=', '>=', '!=', '==']

for op in comp_opers:
    SemanticCube[type_dict['int']][type_dict['int']][operator_dict[op]] = 'yesno'
    SemanticCube[type_dict['dec']][type_dict['dec']][operator_dict[op]] = 'yesno'
    SemanticCube[type_dict['dec']][type_dict['int']][operator_dict[op]] = 'yesno'
    SemanticCube[type_dict['int']][type_dict['dec']][operator_dict[op]] = 'yesno'

for t in type_dict:
    SemanticCube[type_dict[t]][type_dict[t]][operator_dict['=']] = t
SemanticCube[type_dict['dec']][type_dict['int']][operator_dict['=']] = 'dec'

# OPTIONAL
SemanticCube[type_dict['string']][type_dict['string']][operator_dict['+']] = 'string'

for op in ['!=', '==']:
    SemanticCube[type_dict['string']][type_dict['string']][operator_dict[op]] = 'yesno'
    SemanticCube[type_dict['yesno']][type_dict['yesno']][operator_dict[op]] = 'yesno'
```

Descripción detallada del proceso de Administración de Memoria usado en la compilación.

```
class Var:
    def __init__(self, name, type, value, virt_address):
        self.name = name
        self.type = type
        self.value = value
        self.virt_address = virt_address
        self.size = 1

    def print_var(self):
        print("Name", self.name, "Type", self.type, "Virt_Address", self.virt_address,
"Value", self.value)
```

```
from memory_map import MemoryMap

class Function:

    def __init__(self, name = "", type = "", virt_address = ""):
        self.name = name
        self.type = type
        self.variables = {}
        self.counter = 0#location of function to store in memory
        self.signature = [] #example of signature: myFunc(int x, dec y) ==> [0, 2]
        self.index = -1
        self.virt_address = virt_address
        self.local_map = MemoryMap() # local var memory map
        self.temp_map = MemoryMap() # local temp var memory map
```

```
class FunctionDirectory:

    ## http://code.activestate.com/recipes/66531/
    __shared_state = {}
    def __init__(self):
        self.__dict__ = self.__shared_state
        self.global_function = Function("globals")
        self.global_function.type = "int"
        self.functions = {}
        self.current_function = 0
```

```
class MemoryMap:

    def __init__(self):
        self.types = {
            'int' : 0,
            'string' : 0,
            'dec' : 0,
            'yesno' : 0,
            'point' : 0,
            'triangle' : 0,
            'circle' : 0,
```

```
            'rectangle' : 0,
            'canvas' : 0,
            'color' : 0,
        }

    def get_types(self):
        return self.types
```

```
CONST_TABLE_BEGIN_FLAG = "BEGINCONSTTABLE"
CONST_SEGMENT    = "c"
TEMP_SEGMENT     = "t"
LOCAL_SEGMENT    = "l"
GLOBAL_SEGMENT   = "g"

class MemoryController:

    temp_memory = MemoryMap()
    local_memory = MemoryMap()
    global_memory = MemoryMap()
    const_memory = ConstantsTable()
```

```
class Quadruple:

    def __init__(self, operator = "", left_op = "", right_op = "", result = ""):
        #TODO CHECK IF FIXABLE
        self.result = result
        self.left_operand = left_op
        self.right_operand = right_op
        self.operator = operator
```

```
class QuadrupleController:
    quad_list = []
    operator_stack = []
    operand_stack = []
    type_stack = []
    jump_stack = []
    avail = 0
    fake_bottom = '('
    quad_counter = 0
    memory_controller = MemoryController()
```

Descripción de la Máquina Virtual

Equipo de cómputo, Lenguaje y Utilerías.

Marca: Apple.
Modelo: MacBook Pro.
Sistema Operativo: OSX.

Lenguaje Utilizado: Python 2.7.
Análisis Léxico y Semántico: PLY.
Libreria de Salida Gráfica: graphics.py

Administración de Memoria en Ejecución

El proceso de ejecución se lleva a cabo a través de la máquina virtual. Esta contiene una combinación de métodos y clases que cumplen cierta función en el proceso de ejecución. Las principales clases son VMManager y ActivationRecord. La cronología de la ejecución es controlada por el método run, que por su parte delega las responsabilidades a VMManager.

```
class ActivationRecord:

    def __init__(self, local_map, temp_map):
        self.local_mem = {}
        self.temp_mem = {}
        self.return_index = -1
        self.return_address = -1
        for key in type_dict:
            self.local_mem[key] = [None] * local_map[key]
            self.temp_mem[key] = [None] * temp_map[key]
```

Class ActivationRecord.
La clase de ActivationRecord representa la memoria local y temporal de una función, método main o del contexto global. En todos los casos, se utiliza la información que fue colocada en el archivo de código objeto para asignar los tamaños de memoria que serán necesarios para su respectiva función, como mostrado en la imagen anterior. Se itera sobre los tamaños de memoria para todos los tipos de variable y se replica el tamaño en el nuevo Activation Record.

Para poder mantener el control sobre la pila de funciones, fue necesario dividir las responsabilidades en dos diferentes pilas. La primera, “curr_stack”, es sobre la cual las operaciones de asignación operan exclusivamente. La segunda, la pila “call_stack” almacena las funciones que están siendo llamadas. Esto es necesario ya que durante el proceso de llamado a una función, el VMManager tiene que tener acceso a la memoria de ambos intercaladamente. Para agregar función, es necesario crear su ActivationRecord cuando la operación “ERA” es llamada por el método run().

```
    def gen_ar(self, func_name):
        func = self.func_dir.get_function(func_name)
        ltypes = func.get_local_map().get_types()
        ttypes = func.get_temp_map().get_types()
        ar = ActivationRecord(ltypes, ttypes)
        self.call_stack.append(ar)
```

El solo agregarla a la pila de llamadas permite que al agregar un parámetro se pueda cambiar entre contextos durante la misma operación.

```
    def add_param(self, var_address, param_address):
        var_val = self.get_val(var_address)
        self.curr_stack.append(self.call_stack[-1])
        self.set_val(param_address, var_val)
        self.curr_stack.pop()
```

Finalmente, una vez que ya se almacenaron los valores en las memorias correctas. Se agrega la función completa a la pila “actual”.

```
    def go_sub(self, ret_index):
        ar = self.call_stack[-1]
        ar.set_return_index(ret_index)
        self.curr_stack.append(ar)
        self.call_stack.pop()
```

El método get_val() de la clase VMManager muestra como es que se puede acceder al valor de una variable o, similarmente en el método set_val(), asignar un valor a una variable. El proceso es el siguiente.

```
    def get_val(self, address):
        if (address[0] == '*'):
            address = str(self.get_val(address[1:]))
        scope = address[0]
        if (scope == 'g'):
            return self.global_mem.get_val(address)
        elif (scope == 'l'):
            return self.curr_stack[-1].get_val(address)
        elif (scope == 't'):
            if (len(self.curr_stack) > 0):
                return self.curr_stack[-1].get_val(address)
            else:
                return self.global_mem.get_val(address)
        elif (scope == 'c'):
            type1 = type_converter[address[1:3]]
            addr = int(address[3:])
            return self.const_table.types[type1][addr]
```

Se descompone la dirección de la variable en tres componentes, la primera es el primer carácter de la dirección y representa el contexta de la variable. El carácter dos y tres representan el tipo de variable, y por último el resto de los caracteres indican el lugar en la memoria específica en la que se encuentra.

Pruebas del Funcionamiento del Lenguaje:

Pruebas de Lenguaje (Ejemplos) :

Se realizaron pruebas que comprueban el correcto manejo de memoria, creacion de codigo intermedio, y cronología de cuádruplos. Un ejemplo sencillo de esto es el segmento pedazo de código.

```
program simple_quad_test

main [
  yesno ab = (1 + 2 > 1) == (4 * 3 < 1000)

  int bc = 2
  int cd = 1
  print(bc)
  if (cd > bc) [
    bc = 10 + 1
    print(bc)
  ] elsif(cd < bc) [
    bc = 20 + 20
    while ( bc < 100 ) [
      bc = bc + 10
      print(bc)
    ]
  ]
]

finish
```

Resultado
2
False
50
60
70
80
90
100

Este archivo al ser compilado produce el siguiente codigo objeto. En la columna del lado izquierdo de muestra la primera parte, que incluye la lista de cuádruplos que genera el código. En la columna del centro se encuentra la segunda parte, que incluye información relevante al directorio de funciones. La máquina virtual crea este directorio que contiene la información básica necesaria para la creación de los Activation Records. La ejecución de este codigo objeto resulta con la secuencia mostrada en la columna derecha.

```
MAIN   1
+ cin0 cin1 tin0
> tin0 cin2 tye0
* cin3 cin4 tin1
< tin1 cin5 tye1
== tye0 tye1 tye2
= tye2  lye0
= cin6  lin0
= cin7  lin1
PRINT   lin0
< lin0 lin1 tye3
PRINT   tye3
> lin1 lin0 tye4
```

```
GOTO   28
< lin1 lin0 tye5
GOTOF tye5  28
+ cin10 cin11 tin3
= tin3  lin0
< lin0 cin12 tye6
GOTOF tye6  28
+ lin0 cin13 tin4
= tin4  lin0
PRINT   lin0
GOTO   22
```

```
BEGINFUNCTIONS
globals int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point':
0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point':
0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}"
main int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point':
0, 'int': 2, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 1}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point':
0, 'int': 5, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 7}"
BEGINCONSTTABLE
"{'color': [], 'canvas': [], 'triangle': [], 'string': [], 'point':
```

GOTOF tye4 18 + cin8 cin9 tin2 = tin2 lin0 PRINT lin0		[], 'int': ['1', '2', '1', '4', '3', '1000', '2', '1', '10', '1', '20', '20', '100', '10'], 'circle': [], 'dec': [], 'rectangle': [], 'yesno': []}"

Las siguientes son pruebas que se realizaron sobre diferentes funciones del lenguaje Canvas que muestran cómo se comporta el lenguaje.

Programa: **fibonacci.cv**

```
program fibonacci

function fib (int arg1) returns int [
    int val1 = 1
    int val2 = 0
    int sol = 0
    if( arg1 == 1)[
        sol = 1
    ]elsif(arg1 == 0)[
        sol = 0
    ]else[
        val1 = fib(arg1 - 1)
        val2 = fib(arg1 - 2)
        sol = val1 + val2
    ]
    return sol
]

main [
    int myVar = fib(21)
    print(myVar)
]

finish
```

Resultado
10946
Successful

MAIN 26 = cin0 lin1 = cin1 lin2 = cin2 lin3 == lin0 cin3 tye0 GOTOF tye0 8 = cin4 lin3 GOTO 11 == lin0 cin5 tye1 GOTOF tye1 12 = cin6 lin3 GOTO 24 ERA fib - lin0 cin7 tin0 PARAM tin0 lin0 GOSUB gin0 1 = gin0 lin1	ERA fib - lin0 cin8 tin1 PARAM tin1 lin0 GOSUB gin0 1 = gin0 lin2 + lin1 lin2 tin2 = tin2 lin3 = lin3 gin0 ENDPROC ERA fib PARAM cin9 lin0 GOSUB gin0 1 = gin0 lin0 PRINT lin0	BEGINFUNCTIONS globals int "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 1, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" fib int "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 4, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 3, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 2}" main int "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 1, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" BEGINCONSTTABLE "{'color': [], 'canvas': [], 'triangle': [], 'string': [], 'point': [], 'int': ['1', '0', '0', '1', '1', '0', '0', '1', '2', '21'], 'circle': [], 'dec': [], 'rectangle': [], 'yesno': []}"

Programa: **factorial.cv**

```
program factorial

function fact (int arg1) returns int [
    int sol = 1
    if (arg1 > 0) [
      sol = fact(arg1-1) * arg1
    ]
    return sol
]

main [
    int myVar = fact(10)
    print(myVar)
]

finish
```

Resultado
3628800
Successful

```
MAIN   12
= cin0  lin1
> lin0 cin1 tye0
GOTOF tye0  10
ERA fact
- lin0 cin2 tin0
PARAM tin0  lin0
GOSUB gin0  1
* gin0 lin0 tin1
= tin1  lin1
= lin1  gin0
ENDPROC
ERA fact
PARAM cin3  lin0
GOSUB gin0  1
= gin0  lin0
PRINT   lin0
```

```
BEGINFUNCTIONS
globals int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 1, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
main int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 1, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
fact int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 2, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 2, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 1}"
BEGINCONSTTABLE
"{'color': [], 'canvas': [], 'triangle': [], 'string': [], 'point': [], 'int': ['1', '0', '1', '10'], 'circle': [],
'dec': [], 'rectangle': [], 'yesno': []}"
```

Programa: **bubblesort.cv**

```
program bubblesort

int arr[10] = 1

function bubbleSort() returns int [
  yesno sorted = no
  int elem = 0
  int hold = -1
  while (sorted == no) [
    sorted = yes
```

```
function modulo(int a, int b) returns int [
  return a - (b * (a/b))
]

main [
    int foo = 0
    string str = "Unsorted"
    print(str)
    while (foo < 10) [
      arr[foo] = modulo(45 * foo, 7)
```

Resultado
"Unsorted"
0
3
6
2
5
1
4

```
    while (elem < 9) [
      if (arr[elem] > arr[elem + 1])
[
        sorted = no
        hold = arr[elem + 1]
        arr[elem + 1] = arr[elem]
        arr[elem] = hold
      ]
      elem = elem + 1
    ]
    elem = 0
  ]
  return 0
]
```

```
      print(arr[foo])
      foo = foo + 1
    ]
    str = "Sorted"
    print(str)
    foo = bubbleSort()
    while (foo < 10) [
      print(arr[foo])
      foo = foo + 1
    ]
]

finish
```

```
0
3
6
"Sorted"
0
0
1
2
3
3
4
5
6
6
Successfu
l
```

```
MAIN   43
= cin0  gin0
= cye0  lye0
= cin1  lin0
= cin2  lin1
== lye0 cye1 tye0
GOTOF tye0  36
= cye2  lye0
< lin0 cin3 tye1
GOTOF tye1  34
VER  lin0 10
ADDBASE lin0 gin0 *tin0
+ lin0 cin4 tin1
VER  tin1 10
ADDBASE tin1 gin0 *tin2
> *tin0 *tin2 tye2
GOTOF tye2  31
= cye3  lye0
+ lin0 cin5 tin3
VER  tin3 10
ADDBASE tin3 gin0 *tin4
= *tin4  lin1
+ lin0 cin6 tin5
VER  tin5 10
ADDBASE tin5 gin0 *tin6
VER  lin0 10
ADDBASE lin0 gin0 *tin7
= *tin7  *tin6
MAIN   43
= cin0  gin0
= cye0  lye0
= cin1  lin0
= cin2  lin1
== lye0 cye1 tye0
GOTOF tye0  36
= cye2  lye0
< lin0 cin3 tye1
GOTOF tye1  34
VER  lin0 10
ADDBASE lin0 gin0 *tin0
+ lin0 cin4 tin1
VER  tin1 10
```

```
VER  tin5 10
ADDBASE tin5 gin0 *tin6
VER  lin0 10
ADDBASE lin0 gin0 *tin7
= *tin7  *tin6
VER  lin0 10
ADDBASE lin0 gin0 *tin8
= lin1  *tin8
+ lin0 cin7 tin9
= tin9  lin0
GOTO   8
= cin8  lin0
GOTO   5
= cin9  gin10
ENDPROC
/ lin0 lin1 tin0
* lin1 tin0 tin1
- lin0 tin1 tin2
= tin2  gin12
ENDPROC
= cin10  lin0
= cst0  lst0
PRINT   lst0
< lin0 cin11 tye0
GOTOF tye0  62
VER  lin0 10
ADDBASE lin0 gin0 *tin0
ERA modulo
* cin12 lin0 tin1
PARAM tin1  lin0
PARAM cin13  lin1
GOSUB gin12  38
= gin12  *tin0
VER  lin0 10
ADDBASE lin0 gin0 *tin2
PRINT   *tin2
+ lin0 cin14 tin3
= tin3  lin0
GOTO   46
= cst1  lst0
PRINT   lst0
ERA bubbleSort
```

```
BEGINFUNCTIONS
globals int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 15, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 0}"
bubbleSort int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 2, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 1}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 10, 'circle': 0, 'dec': 0,
'rectangle': 0, 'yesno': 3}"
main int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 1,
'point': 0, 'int': 1, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 6, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 2}"
modulo int
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 2, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 0}"
"{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0,
'point': 0, 'int': 3, 'circle': 0, 'dec': 0, 'rectangle':
0, 'yesno': 0}"
BEGINCONSTTABLE
"{'color': [], 'canvas': [], 'triangle': [], 'string':
[""Unsorted"", ""Sorted""], 'point': [], 'int': ['1',
'0', -1, '9', '1', '1', '1', '1', '0', '0', '0', '10', '45', '7',
'1', '10', '1'], 'circle': [], 'dec': [], 'rectangle': [],
'yesno': ['no', 'no', 'yes', 'no']}"
```

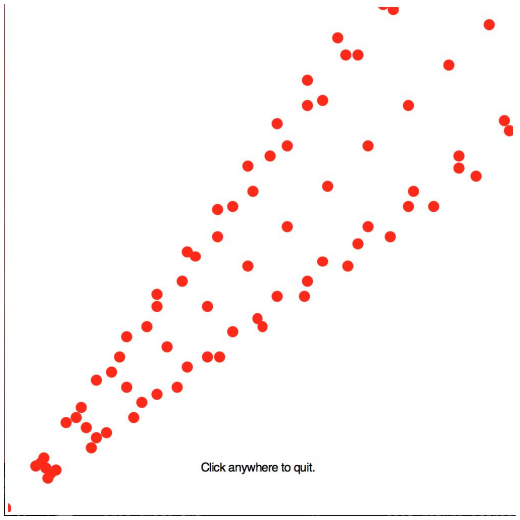
ADDBASE tin1 gin0 *tin2 > *tin0 *tin2 tye2 GOTOF tye2 31 = cye3 lye0 + lin0 cin5 tin3 VER tin3 10 ADDBASE tin3 gin0 *tin4 = *tin4 lin1 + lin0 cin6 tin5	GOSUB gin10 2 = gin10 lin0 < lin0 cin15 tye1 GOTOF tye1 75 VER lin0 10 ADDBASE lin0 gin0 *tin4 PRINT *tin4 + lin0 cin16 tin5 = tin5 lin0 GOTO 67	

Este último programa hace uso de la salida gráfica del lenguaje canvas. A través de la librería graphics.py, se pueden crear gráficas con estatutos integrados al lenguaje como el siguiente llamado circle.

Programa: **graphics_circle.cv**

```
program hello_world2

main [
    canvas myCanvas
    int x_cord = 0
    while (x_cord < 500) [
      circle myCirc (x_cord, x_cord)
      circle myCirc (x_cord - (x_cord/4), x_cord+ (x_cord/16))
      circle myCirc (x_cord + (x_cord/4), x_cord- (x_cord/16))
      circle myCirc (x_cord - (x_cord/8), x_cord+ (x_cord/8))
      circle myCirc (x_cord + (x_cord/8), x_cord- (x_cord/8))
      circle myCirc (x_cord - (x_cord/16), x_cord+ (x_cord/4))
      circle myCirc (x_cord + (x_cord/16), x_cord- (x_cord/4))
      x_cord = x_cord + 40
    ]
    paint myCanvas
]

finish
```

Resultado



MAIN 1 CANVAS = cin0 lin0 < lin0 cin1 tye0 GOTOF tye0 39 CIRCLE lci0 lin0 lin0 / lin0 cin2 tin0 - lin0 tin0 tin1 / lin0 cin3 tin2 + lin0 tin2 tin3 CIRCLE lci1 tin1 tin3 / lin0 cin4 tin4 + lin0 tin4 tin5 / lin0 cin5 tin6 - lin0 tin6 tin7 CIRCLE lci2 tin5 tin7 / lin0 cin6 tin8 - lin0 tin8 tin9 / lin0 cin7 tin10 + lin0 tin10 tin11	CIRCLE lci3 tin9 tin11 / lin0 cin8 tin12 + lin0 tin12 tin13 / lin0 cin9 tin14 - lin0 tin14 tin15 CIRCLE lci4 tin13 tin15 / lin0 cin10 tin16 - lin0 tin16 tin17 / lin0 cin11 tin18 + lin0 tin18 tin19 CIRCLE lci5 tin17 tin19 / lin0 cin12 tin20 + lin0 tin20 tin21 / lin0 cin13 tin22 - lin0 tin22 tin23 CIRCLE lci6 tin21 tin23 + lin0 cin14 tin24 = tin24 lin0 GOTO 3 PAINT	BEGINFUNCTIONS globals int "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 0, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 0}" main int "{'color': 0, 'canvas': 1, 'triangle': 0, 'string': 0, 'point': 0, 'int': 1, 'circle': 7, 'dec': 0, 'rectangle': 0, 'yesno': 0}" "{'color': 0, 'canvas': 0, 'triangle': 0, 'string': 0, 'point': 0, 'int': 25, 'circle': 0, 'dec': 0, 'rectangle': 0, 'yesno': 1}" BEGINCONSTTABLE "{'color': [], 'canvas': [], 'triangle': [], 'string': [], 'point': [], 'int': ['0', '500', '4', '16', '4', '16', '8', '8', '8', '8', '16', '4', '16', '4', '40'], 'circle': [], 'dec': [], 'rectangle': [], 'yesno': []}"

Listados Perfectamente Documentados del Proyecto

Lista de Modulos

Graphics.py	Librería orientada a objetos con output gráfico. Diseñada para novatos en la programación, facilita la utilización de la librería de Tinker de Python para la creacion de graficos. Como
Scanner	El Scanner es el analizador léxico del lenguaje. Identifica cada ‘token’ en el programa y se lo envía al Parser para continuar el proceso de compilación. Es en esta etapa donde verifica que las palabras clave del

	lenguaje están siendo usadas correctamente.
Parser	Gramática del lenguaje de programación Canvas. Tiene como entrada tokens enviados por el analizador léxico en el archivo Scanner. Como salida, el Parser genera errores cuando estos aparecen en compilación y, si no es el caso, valida que la sintaxis del programa sea la correcta. Parser utiliza métodos de QuadController para generar cuadruplos. También tiene acceso al MemoryController, que utiliza para generar las direcciones de memoria que se asignan a las variables.
FunctionDirectory	Directorio de funciones que mantiene registro de qué funciones están siendo declaradas durante compilación. Cada función tiene la información de su firma, el tipo de retorno, y número de espacios en memoria necesarios para su instanciación.
QuadController	Controlador que se encarga del manejo de cuádruplos durante compilación. Este controlador maneja las pilas de operandos y operadores y se encarga, al igual que el parser, de verificar la semántica del programa. Utiliza la pila de tipos para comparar operandos a lo largo de las expresiones.
MemoryController	Controlador que se encarga de incrementar los contadores de memoria dentro del scope correspondiente, dependiendo la cantidad de declaraciones que vaya leyendo.
ActivationRecord	Memoria en ejecución que guarda los espacios de memoria necesarios para cada función que fue llamada.
VMManager	Módulo que se encarga del proceso de ejecución del código objeto y asignación y extracción de valores en memoria. El VMManager mantiene dos pilas, una de funciones y otra de funciones siendo llamadas. La primera es donde se mantiene un registro de en qué nivel de funciones se encuentra la ejecución, la segunda se utiliza para poder ingresar los valores de una función a otra durante la instanciación.

Manual de Usuario - Canvas Quick Reference

El siguiente es el contenido del manual de usuario.

1. Creando un programa
2. Expresiones aritméticas, lógicas y relacionales
3. Impresión
4. Control de flujo
5. Cambio de contexto
6. Elementos no atómicos - Output gráfico

1. Creando un programa

Todos los programas de Canvas siguen la siguiente estructura

```
program hello_world2

main [

]

finish
```

Se comienza con la palabra reservada “program” seguida del nombre de tu elección para tu programa. En este caso se utiliza *hello_world2* como el nombre del programa.
Seguido se escribe la función principal *main* y su bloque correspondiente.
Finalmente se termina el programa escribiendo la palabra *finish*.

2. Expresiones aritméticas

Declaración de una variable

```
program hello_world2

main [

   int myVar = 100

]

finish
```

Para declarar una variable se comienza escribiendo su tipo (en este caso es *int*), seguido de un signo de “=” y un valor que coincida con el tipo con el que la variable fue declarado.

Expresiones aritméticas

```
program hello_world2
```

```
main [

   int foo = 100
   int bar = 420
   int baz = 320

   baz = bar + foo
   foo = bar - foo


]

finish
```

Después de declarar unas variables, hacemos algunas expresiones aritméticas. La sintaxis es natural pues es semejante al formato que hemos conocido todas nuestras vidas.

3. Impresión

Impresión a línea de comandos

```
program hello_world2

main [

   int foo = 1
   print(foo)

]

finish
```

4. Control de flujo

```
program hello_world2

main [
  yesno ab = (1 + 2 > 1) == (4 * 3 < 1000)

  int bc = 2
  int cd = 1
  print(bc)
  if (cd > bc) [
    bc = 10 + 1
    print(bc)
  ] elsif(cd < bc) [
    bc = 20 + 20
    while ( bc < 100 ) [
```

```
      bc = bc + 10
      print(bc)
    ]
  ]
]

finish
```

Para declarar un bucle, utilizar la palabra *while* seguida de una expresión entre paréntesis y después un bloque. Para tomar decisiones, utilizar *if* para la primera expresión, *elsif* para siguientes, terminando con *else*. El *else* es **obligatorio** si se utilizó elsif.

5. Cambio de contexto

```
program factorial

function fact (int arg1) returns int [
    int sol = 1
    if (arg1 > 0) [
      sol = fact(arg1-1) * arg1
    ]
    return sol
]

main [
    int myVar = fact(10)
    print(myVar)
]
```

Para declarar una nueva función, utilizar la sintax: *function* id (params) *returns* tipo_retorno bloque.

6. Elementos no atómicos - arreglos

```
program hello_world2

main [
    int arr[3] = 0
    int foo = 1
    arr[2] = 3 + 4
]

finish
```

Para declarar una nueva arreglo, utilizar los ‘[]’ después de el nombre de la variable a declarar para inicializar su tamaño. Para accesar el subíndice, llamar al arreglo con una expresión que represente un subíndice **menor** a su tamaño originalmente inicializado.

7. Output gráfico

```
program hello_world2

main [
    canvas myCanvas
    int x_cord = 0
    while (x_cord < 500) [
      circle myCirc (x_cord, x_cord)
      circle myCirc (x_cord - (x_cord/4), x_cord+ (x_cord/16))
      circle myCirc (x_cord + (x_cord/4), x_cord- (x_cord/16))
      circle myCirc (x_cord - (x_cord/8), x_cord+ (x_cord/8))
      circle myCirc (x_cord + (x_cord/8), x_cord- (x_cord/8))
      circle myCirc (x_cord - (x_cord/16), x_cord+ (x_cord/4))
      circle myCirc (x_cord + (x_cord/16), x_cord- (x_cord/4))
      x_cord = x_cord + 40
    ]
    paint myCanvas
]

finish
```

Inicializar una variable canvas para utilizar una ventana gráfica. Utilizar *circle(x, y)* o *rectangle(x, y)* para crear un elemento gráfico a imprimir en pantalla. Finalmente pintar en pantalla utilizando el comando *paint* seguido del nombre de la variable de *canvas* anteriormente declarada.