## Descripción del proyecto

### Propósito, Objetivos y alcance del proyecto

El lenguaje ***Objective uwu*** tiene como propósito el hacer que los niños entiendan la lógica básica para programación. Se cumple este propósito en un ambiente estilo juego, donde el niño puede experimentar comandos para dibujar en el ambiente gráfico del compilador.

El objetivo principal del lenguaje ***Objective uwu*** es controlar a Kame-chan mediante comandos dentro del ambiente gráfico, con el objetivo de que los niños vean como Kame-chan se mueve basado a lo que ellos programan para que logren mover a Kame-chan a la meta.

El compilador propuesto será capaz de leer y procesar operaciones aritméticas en forma de instrucciones para Kame-chan, así como ciclos y otras funciones con el fin de mostrar en pantalla los movimientos de Kame-chan basados en el código que se esté implementando.

### Análisis de Requerimientos y Casos de Uso generales.

Los requerimientos considerados para el desarrollo del proyecto son los siguientes:

* El lenguaje debe de aceptar números enteros así como decimales
* El lenguaje debe manejar ciclos simples así como estatutos condicionales
* El lenguaje debe de aceptar declaraciones de variables de los tipos enteras, booleanas y flotantes
* El lenguaje debe ser capaz de manejar las funciones de tipo de retorno void, entero, flotante
* El lenguaje debe de poder manejar recursión en funciones
* El lenguaje acepta declaración de arreglos de 1 o más dimensiones
* El lenguaje muestra los movimientos de kame chan de acuerdo a lo programado por el usuario

### Descripción de los principales Test Cases.

**Factorial recursivo**

Usando una función que regresa una llamada recursiva calcular el factorial del número n dado a la función.

**Factorial cíclico**

Usando ciclos calcular el factorial del número n.

**Fibonacci recursivo**

Usando una función que regresa una llamada recursiva calcular el número fibonacci del número n dado a la función.

**Fibonacci cíclico**

Usando ciclos calcular la posición n de la serie fibonacci.

**Sort de un arreglo**

Ordenar un arreglo de menor a mayor o de mayor a menor utilizando algún algoritmo de ordenamiento

**Find en un arreglo**

Encontrar la posición de cierto valor dentro de un arreglo.

**Cambio de color, grosor del rastro que dibuja Kame-chan**

Ser capaz de cambiar el color y grosor de la línea que dibuja Kame-chan sin problemas.

**Triángulo de Sierpinski**

Ser capaces de dibujar el triángulo de sierpinski recursivamente

**Dibujo de objetos especificados(Cuadro, circulo)**

Kame-chan debe de ser capaz de dibujar estos objetos automáticamente cuando se llame a la función predeterminada para cada objeto.

### Descripción del PROCESO general seguido para el desarrollo del proyecto, incluyendo Bitácoras generales y un pequeño párrafo de reflexión de cada alumno, en relación a los principales aprendizajes logrados (firmarlo).

El proceso de desarrollo del proyecto fue juntarnos cada semana el sábado para hacer el avance correspondiente a la semana y subirlo ese mismo día, en la mayor parte nuestro proyecto siempre fue al corriente con las entregas que eran sugeridas hasta las últimas entregas cambiamos un poco las cosas debido a que la dificultad de las tareas aumentaba empezamos a juntarnos viernes y sábado para hacer las entregas correspondientes y hacer los testeos que correspondan para verificar que el funcionamiento del programa siguiera de la forma normal y esperada.

**Bitácora de avance 1, Febrero 26 a Marzo 2**

Realizamos lo que es el analizador de lexico y sintactico o lexer y parser basándonos en lo que fue el lexer y parser del lenguaje patito usando python y PLY, también declaramos las palabras reservadas así como los tokens que usamos.

**Bitácora de avance 2, Marzo 5 - 9**

Realizamos el directorio de variables y de procedimientos así como nuestro cubo semántico para poder checar validaciones de tipos.

**Bitácora de avance 3, Marzo 12 - 16**

Terminamos la implementación del cubo semántico, también realizamos la generación de código para expresiones aritméticas y estatutos de condiciones y asignaciones, además de esto realizamos pruebas a todos estos cambios con archivos de texto específicos.

**Bitácora de avance 4, Marzo 19 - 23**

Realizamos generacion de codigo de estatutos condicionales así como decisiones y ciclos, se probó con diferentes casos y se encontraron bugs en el ciclo, el ciclo se hacia 1 vez meno de lo esperado y se soluciono en ese mismo momento ya que el error era un índice inicializado mal

**Bitácora de avance 5, Abril 2 - 6**

Realizamos la implementación de código de generación de funciones para los diferentes tipos de funciones que se pueden declarar en el programa. se hicieron pruebas con diferentes tipos de funciones declaradas

**Bitácora de avance 6, Abril 9 - 13:**

Realizamos el mapeo inicial de memoria para la aplicación así como la máquina virtual de estatutos secuenciales y la máquina virtual de expresiones aritméticas, se realizaron pruebas en estas y no obtuvimos ningún tipo de error cuando se esperaba un resultado correcto.

**Bitácora de avance 7, Abril 16 - 20**

Se comenzó a trabajar sobre arreglos y sus implementaciones en memoria, maquina virtual y generacion de codigo.

Párrafo de reflexión Roger

Párrafo de reflexión Alejandro

## Descripción del lenguaje

### Nombre del lenguaje

**Objective-uwu**

### Descripción de las principales características del lenguaje

Objective-uwu es un lenguaje de output gráfico, los usuarios crean codigo y lo codificado es interpretado y ejecutado por Kame-chan dentro de un ambiente gráfico delimitado. El lenguaje es capaz de manejar funciones con valores de retorno, funciones void, funciones recursivas, así como declarar arreglos, uso de estatutos condicionales y ciclos así como uso de variables globales y locales.

### Listado de errores que pueden ocurrir en ejecución y compilación

* Declarar variables que ya existentes dentro del programa
* Asignación de valores no compatibles con los tipos de la variable definida
* Intentar usar una variable no definida
* Sintaxis del programa mal escrita
* Declarar nuevamente una función ya declarada
* Llamar a una función que no ha sido declarada
* El arreglo que se intenta acceder no ha sido declarado
* Número de parámetros diferentes a los esperados por la función
* Parámetro no existente en función llamada
* Error de tipo al enviar parametro
* Error de tipo al retornar en la función
* Errores de tipo en operaciones aritméticas

## **Descripción del compilador**

### Equipo de computo, lenguaje y utilerías utilizadas en el desarrollo del proyecto

Para el desarrollo del proyecto utilizamos nuestras computadoras personales, utilizamos Python 2.7 para la realización del parser y el scanner. de igual forma utilizamos github para mantener control de versiones y ir subiendo los cambios a los diferentes archivos.

### Descripción de análisis de léxico

A continuación se presentan el léxico del lenguaje así como sus tokens y palabras reservadas

**Tokens**

|  |  |  |
| --- | --- | --- |
| digit | [0-9] | identificador para dígitos del 0 al 9 |
| Num | digit+ | Una cadena de dígitos |
| Flot | Num\.Num | Una cadena de dígitos que se separan por un punto decimal seguido de más dígitos |
| lwc | [a-z] | letras minúsculas del alfabeto |
| upc | [A-Z] | letras mayúsculas del alfabeto |
| id | lwc(lwc|upc|dig)\* | combinación de letras minúsculas, mayúsculas y dígitos para hacer identificadores |
| opmat | \+| - | \\* | / | % | Operadores aritméticos |
| oprel | <|>|<=|>=|==|!= | Operadores lógicos |
| oplog | AND|OR|NOT | Operadores lógicos |

**Palabras reservadas**

|  |  |
| --- | --- |
| PROG | Marca inicio de programa |
| FUNC | Marca inicio de función |
| RET | Regresa valor |
| MAIN | Marca inicio de módulo principal |
| KAMEF | Movimiento hacia adelante |
| KAMEB | Movimiento hacia atrás |
| KAMER | Rotación |
| DRAW | Dibuja línea |
| CIRCLE | Dibuja círculo |
| SQUARE | Dibuja cuadrado |
| IF | Condicional |
| ELSE | Default condicional |
| LOOP | Ciclo finito |
| ITER | Iteración de ciclo |
| TRUE | Verdadero |
| FALSE | Falso |

**Operadores lógicos**

|  |  |
| --- | --- |
| NOT | Negación |
| AND | Intersección |
| OR | Unión |

**Operadores aritméticos**

|  |  |
| --- | --- |
| + | Suma |
| - | Resta |
| \* | Multiplicación |
| / | División |
| % | Módulo |

**Operadores relacionales**

|  |  |
| --- | --- |
| == | Igual a |
| != | Diferente de |
| < | Menor que |
| <= | Menor o igual que |
| > | Mayor que |
| >= | Mayor o igual que |

### Descripción de análisis de sintaxis

A continuación se presenta nuestra gramatica completa en forma de expresiones regulares:

prog → PROG { declare mainBlock }

val → [0-9]+ | [0-9]+ ‘.’ [0-9]+ | TRUE | FALSE |[a-zA-Z][a-zA-Z0-9]\*

declare → decVar decFunc

decVar → var decVar | ‘ ’

decFunc → func decFunc | ‘ ‘

var → VAR tipo [a-zA-Z][a-zA-Z0-9]\* arrayCreate

arrayCreate → [ [0-9]+ ] | [ [0-9]+ ] [ [0-9]+ ] | ‘ ‘

arrayIndex → [ exp ] | [ exp ] [ exp ] | ‘ ‘

tipo → NUM | FLOT | BOOL

assign → assignTo = megaExp

assignTo → [a-zA-Z][a-zA-Z0-9]\* arrayIndex | KEMEF | KAMER| KAMEB

func → func1 func2

func1 → func11 func12

func11 → FUNC decideType [a-zA-Z][a-zA-Z0-9]\* (

func12 → params ) {

func2 → decVar bloque }

decidetype → tipo | VOID

params → tipo [a-zA-Z][a-zA-Z0-9]\* moreParams | ‘ ‘

moreParams → ‘,’ tipo [a-zA-Z][a-zA-Z0-9]\* moreParams

mainBlock → mainBlock1 bloque }

mainBlock1 → MAIN {

opLogico → AND | OR

loop → loop1 loop2 loop3

loop1 → LOOP

loop2 → ( exp )

loop3 → { bloque }

opRelacional → == | != | < | <= | > | >=

bloque → estructura bloque | #uwu

estructura → assign | loop | comparacion | return | funcCall | decVar

funcCall → funcCall1 funcCall2 | DRAW ( megaExp ) | CIRCLE ( exp ) | SQUARE ( exp ) | SIZE ( exp )

| COLOR ( colorChoice )

funcCall1 → [a-zA-Z][a-zA-Z0-9]\* (

funcCall2 → paramVals )

bool → TRUE | FALSE

colorChoice → RED|GREEEN|BLUE|VIOLET|ORANGE|WHITE|BLACK

paramVals → unParam moreParamVals

moreParamVals → ‘,’ unParam moreParamVals | ‘,’

unParam → [a-zA-Z][a-zA-Z0-9]\* ‘:’ megaExp

return → RET megaExp

comparacion → compara1 compara2

compara1 → IF ‘(‘ megaExp ‘)’ {

compara2 → bloque } maybeElse

maybeEls → checkElse doElse | ‘ ‘

checkElse → ELSE {

doElse → bloque }

megaExp → maybeNot superExp anotherMega

maybeNot → NOT | ‘ ‘

anotherMega → opLogico megaExp | ‘ ‘

superExp → exp maybeRel

maybeRel → opRelacional exp | ‘ ‘

exp → term anotherExp

anotherExp → ‘+’ exp | ‘-’ exp | ‘ ‘

term → fact anotherTerm

anotherTerm → ‘\*’ term | ‘/’ term | % term | ‘ ‘

fact → ‘(‘ megaExp ‘)’ | [a-zA-Z][a-zA-Z0-9]\* arrayIndex | funcCall | val

### Descripción de generación de código intermedio y análisis semántico

### Codigos de operacion del lenguaje

Para el código intermedio utilizamos cuadruplos parecidos a los vistos en clase, además de operaciones aritméticas, lógicas y booleanas utilizamos los siguientes códigos de operación dentro del lenguaje objective-uwu:

|  |  |
| --- | --- |
| GOTO | Salto a cierto cuádruplo |
| GOTOF | Salto a cierto cuádruplo cuando una condición no se cumple |
| ERA | Guarda en la memoria el registro de activación de una función |
| GOSUB | Salto a cierta subrutina en el código |
| PARAM | Código para los parámetros de cierta función |
| RET | Código para retornar un valor de una función |
| ENDPROC | Código que marca el final de una función |

### Diagramas de sintaxis con acciones correspondientes

### Acciones semánticas

Para el caso de nuestra semántica se hizo un cubo con todas las combinaciones válidas posibles y sus resultados, además de esto se hizo una función que checara si los 3 parámetros que se le enviaban al cubo existían dentro de él, de lo contrario se retornaba error, a continuación se muestra el cubo semántico en la forma de Operador Operando Operador y resultado.

**Cubo semántico**

|  |  |  |  |
| --- | --- | --- | --- |
| NUM | + | NUM | NUM |
| NUM | \* | NUM | NUM |
| NUM | / | NUM | FLOT |
| NUM | % | NUM | NUM |
| NUM | < | NUM | BOOL |
| NUM | > | NUM | BOOL |
| NUM | <= | NUM | BOOL |
| NUM | >= | NUM | BOOL |
| NUM | != | NUM | BOOL |
| NUM | == | NUM | BOOL |
| NUM | = | NUM | NUM |
| NUM | + | FLOT | FLOT |
| NUM | \* | FLOT | FLOT |
| NUM | / | FLOT | FLOT |
| NUM | < | FLOT | BOOL |
| NUM | > | FLOT | BOOL |
| NUM | <= | FLOT | BOOL |
| NUM | >= | FLOT | BOOL |
| NUM | != | FLOT | BOOL |
| NUM | == | FLOT | BOOL |
| NUM | = | FLOT | NUM |
| FLOT | + | FLOT | FLOT |
| FLOT | \* | FLOT | FLOT |
| FLOT | - | FLOT | FLOT |
| FLOT | / | FLOT | FLOT |
| FLOT | < | FLOT | BOOL |
| FLOT | > | FLOT | BOOL |
| FLOT | <= | FLOT | BOOL |
| FLOT | >= | FLOT | BOOL |
| FLOT | != | FLOT | BOOL |
| FLOT | == | FLOT | BOOL |
| FLOT | = | FLOT | BOOL |
| BOOL | AND | BOOL | BOOL |
| BOOL | OR | BOOL | BOOL |
| BOOL | = | BOOL | BOOL |

### Descripción detallada del proceso de administración de memoria

Para el manejo de memoria tenemos un diccionario que empieza vacío llamado memoria, además de este tenemos un diccionario que adentro contiene las direcciones de memoria iniciales para las variables globales y las variables temporales, esto con el fin de ser accesadas y posteriormente agregadas con esa dirección al diccionario de memoria.

DIBUJO A MANO DE COMO SE VE LA MEMORIA CON VALORES para poder explicarla

## Descripción de la máquina virtual

### Descripción detallada del proceso de administración de memoria en ejecución

## Pruebas del funcionamiento del lenguaje (Código y resultados arrojados por generación de código y ejecución)

**Factorial recursivo**

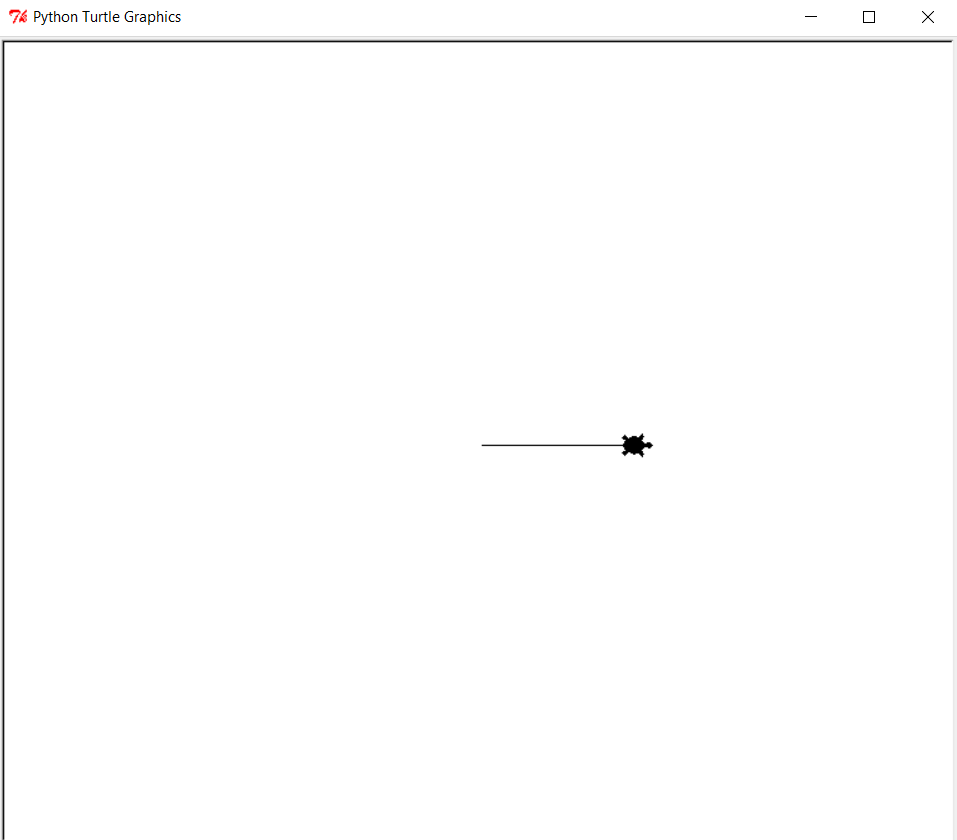
PROG{  
 FUNC NUM fact(NUM n){  
 IF(n<=1){  
 RET 1  
 #uwu  
 }  
 ELSE{  
 RET fact(n:n-1)\*n  
 #uwu  
 }  
 #uwu  
 }  
 MAIN{  
 KAMEF = fact(n:5)  
 #uwu  
 }  
}

Resultados de generacion de codigo

{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 13}  
{'operator': '<=', 'rightOperand': 1, 'leftOperand': 'n', 'result': '(25000)'}  
{'operator': 'GOTOF', 'rightOperand': '', 'leftOperand': '(25000)', 'result': 5}  
{'operator': 'RET', 'rightOperand': 1, 'leftOperand': '', 'result': ''}  
{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 12}  
{'operator': 'ERA', 'rightOperand': 'fact', 'leftOperand': '', 'result': ''}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'n', 'result': '(15000)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(15000)', 'result': 'fact:n'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'fact', 'result': ''}  
{'operator': '=', 'rightOperand': 'fact', 'leftOperand': '', 'result': '(15001)'}  
{'operator': '\*', 'rightOperand': 'n', 'leftOperand': '(15001)', 'result': '(15002)'}  
{'operator': 'RET', 'rightOperand': '(15002)', 'leftOperand': '', 'result': ''}  
{'operator': 'ENDPROC', 'rightOperand': '', 'leftOperand': '', 'result': ''}  
{'operator': 'ERA', 'rightOperand': 'fact', 'leftOperand': '', 'result': ''}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': 5, 'result': 'fact:n'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'fact', 'result': ''}  
{'operator': '=', 'rightOperand': 'fact', 'leftOperand': '', 'result': '(15003)'}  
{'operator': 'KAMEF', 'rightOperand': '(15003)', 'leftOperand': '', 'result': ''}

Resultado arrojado por ejecución

15003:120



**Factorial cíclico**

**Fibonacci recursivo**

**Fibonacci cíclico**

PROG{

VAR NUM num1

VAR NUM t1

VAR NUM t2

VAR NUM nextTerm

VAR NUM resp

MAIN{

COLOR(GREEN)

t1 = 0

t2 = 1

num1 = 6

IF(num1 == 0)

{

resp = t1

#uwu

}

ELSE{

resp = t2

#uwu

}

LOOP(num1-1)

{

nextTerm = t1 + t2

t1 = t2

t2 = nextTerm

resp = t2

KAMEF = 20

CIRCLE(resp\*2)

#uwu

}

#uwu

}

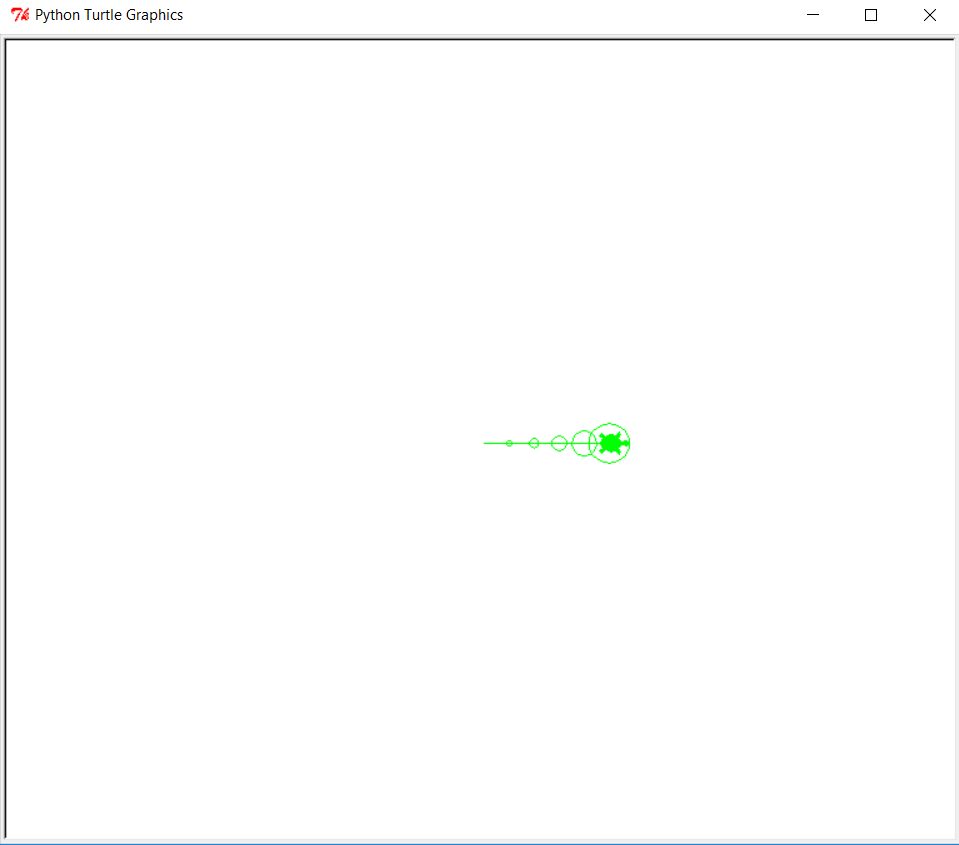
}

Resultados de generacion de codigo

{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 1}  
{'operator': 'COLOR', 'rightOperand': '', 'leftOperand': '', 'result': 'GREEN'}  
{'operator': '=', 'rightOperand': 0, 'leftOperand': '', 'result': 't1'}  
{'operator': '=', 'rightOperand': 1, 'leftOperand': '', 'result': 't2'}  
{'operator': '=', 'rightOperand': 6, 'leftOperand': '', 'result': 'num1'}  
{'operator': '==', 'rightOperand': 0, 'leftOperand': 'num1', 'result': '(25000)'}  
{'operator': 'GOTOF', 'rightOperand': '', 'leftOperand': '(25000)', 'result': 9}  
{'operator': '=', 'rightOperand': 't1', 'leftOperand': '', 'result': 'resp'}  
{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 10}  
{'operator': '=', 'rightOperand': 't2', 'leftOperand': '', 'result': 'resp'}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'num1', 'result': '(15001)'}  
{'operator': '<=', 'rightOperand': '(15001)', 'leftOperand': '(15000)', 'result': '(25001)'}  
{'operator': 'GOTOF', 'rightOperand': '', 'leftOperand': '(25001)', 'result': 23}  
{'operator': '+', 'rightOperand': 't2', 'leftOperand': 't1', 'result': '(15002)'}  
{'operator': '=', 'rightOperand': '(15002)', 'leftOperand': '', 'result': 'nextTerm'}  
{'operator': '=', 'rightOperand': 't2', 'leftOperand': '', 'result': 't1'}  
{'operator': '=', 'rightOperand': 'nextTerm', 'leftOperand': '', 'result': 't2'}  
{'operator': '=', 'rightOperand': 't2', 'leftOperand': '', 'result': 'resp'}  
{'operator': 'KAMEF', 'rightOperand': 20, 'leftOperand': '', 'result': ''}  
{'operator': '\*', 'rightOperand': 2, 'leftOperand': 'resp', 'result': '(15003)'}  
{'operator': 'CIRCLE', 'rightOperand': '(15003)', 'leftOperand': '', 'result': ''}  
{'operator': '+', 'rightOperand': 1, 'leftOperand': '(15000)', 'result': '(15000)'}  
{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 10}  
{'operator': '=', 'rightOperand': 1, 'leftOperand': '', 'result': '(15000)'}

Resultado arrojado por ejecución

15002:8



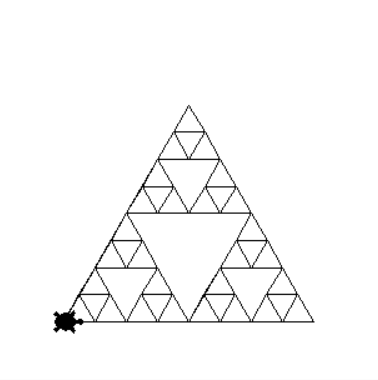
**Triángulo de Sierpinski**

PROG{  
 FUNC VOID sierpinski(NUM largo, NUM deep){  
  
 IF(deep == 0)  
 {  
 VAR NUM range  
 range = 5  
 LOOP(range)  
 {  
 KAMEF = largo  
 KAMER = 120  
 range = range - 1  
 #uwu  
 }  
 #uwu  
 }  
 ELSE  
 {  
 sierpinski(largo:largo/2, deep:deep-1)  
 KAMEF = largo/2  
 sierpinski(largo:largo/2, deep:deep-1)  
 KAMEB = largo/2  
 KAMER = 60  
 KAMEF = largo/2  
 KAMER = 300  
 sierpinski(largo:largo/2, deep:deep-1)  
 KAMER = 60  
 KAMEB = largo/2  
 KAMER = 300  
 #uwu  
 }  
 #uwu  
 }  
 MAIN{  
 sierpinski(largo:200, deep:3)  
 #uwu  
  
 }  
}

Resultados de generacion de codigo:

{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 45}  
{'operator': '==', 'rightOperand': 0, 'leftOperand': 'deep', 'result': '(25000)'}  
{'operator': 'GOTOF', 'rightOperand': '', 'leftOperand': '(25000)', 'result': 14}  
{'operator': '=', 'rightOperand': 5, 'leftOperand': '', 'result': 'range'}  
{'operator': '<=', 'rightOperand': 'range', 'leftOperand': '(15000)', 'result': '(25001)'}  
{'operator': 'GOTOF', 'rightOperand': '', 'leftOperand': '(25001)', 'result': 12}  
{'operator': 'KAMEF', 'rightOperand': 'largo', 'leftOperand': '', 'result': ''}  
{'operator': 'KAMER', 'rightOperand': 120, 'leftOperand': '', 'result': ''}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'range', 'result': '(15001)'}  
{'operator': '=', 'rightOperand': '(15001)', 'leftOperand': '', 'result': 'range'}  
{'operator': '+', 'rightOperand': 1, 'leftOperand': '(15000)', 'result': '(15000)'}  
{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 4}  
{'operator': '=', 'rightOperand': 1, 'leftOperand': '', 'result': '(15000)'}  
{'operator': 'GOTO', 'rightOperand': '', 'leftOperand': '', 'result': 44}  
{'operator': 'ERA', 'rightOperand': 'sierpinski', 'leftOperand': '', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20000)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(20000)', 'result': 'sierpinski:largo'}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'deep', 'result': '(15002)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(15002)', 'result': 'sierpinski:deep'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'sierpinski', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20001)'}  
{'operator': 'KAMEF', 'rightOperand': '(20001)', 'leftOperand': '', 'result': ''}  
{'operator': 'ERA', 'rightOperand': 'sierpinski', 'leftOperand': '', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20002)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(20002)', 'result': 'sierpinski:largo'}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'deep', 'result': '(15003)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(15003)', 'result': 'sierpinski:deep'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'sierpinski', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20003)'}  
{'operator': 'KAMEB', 'rightOperand': '(20003)', 'leftOperand': '', 'result': ''}  
{'operator': 'KAMER', 'rightOperand': 60, 'leftOperand': '', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20004)'}  
{'operator': 'KAMEF', 'rightOperand': '(20004)', 'leftOperand': '', 'result': ''}  
{'operator': 'KAMER', 'rightOperand': 300, 'leftOperand': '', 'result': ''}  
{'operator': 'ERA', 'rightOperand': 'sierpinski', 'leftOperand': '', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20005)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(20005)', 'result': 'sierpinski:largo'}  
{'operator': '-', 'rightOperand': 1, 'leftOperand': 'deep', 'result': '(15004)'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': '(15004)', 'result': 'sierpinski:deep'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'sierpinski', 'result': ''}  
{'operator': 'KAMER', 'rightOperand': 60, 'leftOperand': '', 'result': ''}  
{'operator': '/', 'rightOperand': 2, 'leftOperand': 'largo', 'result': '(20006)'}  
{'operator': 'KAMEB', 'rightOperand': '(20006)', 'leftOperand': '', 'result': ''}  
{'operator': 'KAMER', 'rightOperand': 300, 'leftOperand': '', 'result': ''}  
{'operator': 'ENDPROC', 'rightOperand': '', 'leftOperand': '', 'result': ''}  
{'operator': 'ERA', 'rightOperand': 'sierpinski', 'leftOperand': '', 'result': ''}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': 200, 'result': 'sierpinski:largo'}  
{'operator': 'PARAM', 'rightOperand': '', 'leftOperand': 3, 'result': 'sierpinski:deep'}  
{'operator': 'GOSUB', 'rightOperand': '', 'leftOperand': 'sierpinski', 'result': ''}

Resultado arrojado por ejecución



## Listados perfectamente documentados del proyecto

A continuación describiremos y mostraremos algunos de los listados de códigos que son vitales para que Objective-uwu funcione

**Función semantic\_check**

El semantic check es una parte clave de nuestro lenguaje, este nos permite detectar errores semánticos antes de agregar cuadruplos mediante la búsqueda de la búsqueda de los parámetros de la función en el cubo semántico, nuestro cubo semántico tiene solo las combinaciones correctas de valores y operadores, de lo contrario se regresa error y ya desplegamos que hubo un error.

def semantic\_check(lOP\_type,rOP\_type,oper):

if lOP\_type in sem\_cube:

if rOP\_type in sem\_cube[lOP\_type]:

if oper in sem\_cube[lOP\_type][rOP\_type]:

return sem\_cube[lOP\_type][rOP\_type][oper]

return 'error'

## Manual de usuario quick reference

### Declaraciones y asignaciones

Es posible declarar funciones dentro del scope MAIN o dentro de un scope en específico para que pertenezcan al scope de esa función en específico.

MAIN{

VAR NUM x

x = 6

VAR FLOT y

y = 6.5

VAR BOOL z

z = True

}

### Estatutos condicionales

En nuestro lenguaje cada bloque de estatutos es necesario que termine con #uwu para indicar su fin

MAIN{

IF(x == 6)

{

Haz algo

#uwu

}

ELSE

{

Haz esto

#uwu

}

}

### Ciclos

LOOP( y)

{

Haz algo

#uwu

}

### Funciones y llamadas a funciones

las declaraciones de funciones se hacen de la siguiente forma en Objective-uwu:

FUNC NUM move(NUM x, FLOT y, FLOT z)

{

haz algo

RET x

#uwu

}

y las llamadas a la función se hacen de la siguiente forma:

MAIN{

VAR NUM x

x = move(x:21, z:15+2.5, y:15

#uwu

}

Como podemos ver en la llamada de la función los parámetros no necesariamente deben de estar en orden del que se recibirán, ya que nuestro lenguaje identifica qué valor es de cada parámetro y lo asigna de forma correcta para su uso en la función.

### Output gráfico

Kame-chan tiene diferentes comandos ya predeterminados, como hacer figuras cambiar el color de línea así como el grosor de la línea y escoger si se sigue dibujando o no.

Para el caso del color están disponibles los siguientes colores: Violeta, Verde, Azul,Rojo,Blanco,Negro,Amarillo

se cambia el color con la instrucción:

COLOR(yellow) o COLOR(YELLOW)

esto cambiará el color a amarillo.

Para el tamaño de la brocha es posible hacerla mas grande o más pequeña, no es posible intentar dar un valor negativo en este caso para el tamaño de la brocha:

SIZE(25) o SIZE(2.5)

Esto cambiará el tamaño de la brocha al valor entero o flotante que se le asigne

De igual forma las figuras predeterminadas que puede hacer Kame-Chan aceptan números enteros o flotantes que definirán su largo:

CIRCLE(25) o CIRCLE(2.5)

SQUARE(25) o SQUARE(2.5)