Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Área de Ciencias de la Computación
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2019
Catedráticos: Ing. Manuel Castillo, Ing. Mario Bautista, Ing. Kevin Lajpop
Auxiliares: Nery Gálvez, Miguel Ruano, Erick Tejaxún

Segundo Proyecto de Laboratorio

PyUsac

Descripción

En la actualidad existen diversos lenguajes de programación con diversos paradigmas en su implementación. Cada uno tiene sus ventajas sobre otros y esto nos permite poder elegir entre cuál sea el que más se acople a nuestras necesidades como desarrolladores. Por lo cual el segundo proyecto de laboratorio de programación consta en la construcción de un intérprete de un lenguaje de programación que nos permita ejecutar un lenguaje de programación llamado pyUsac.

Objetivos

General

Que el estudiante pueda aplicar los conocimientos de las fases de análisis léxico, sintáctico y semántico para poder desarrollar un intérprete funcional de un lenguaje de programación.

Específicos

- Que el estudiante sea capaz de utilizar herramientas para generación de analizadores léxicos y sintácticos y pueda aplicarlas a soluciones de sistemas informáticos.
- Que el estudiante sea capaz de realizar un intérprete de un lenguaje de programación utilizando para ello las diferentes fases del proceso de compilación.
- Que el estudiante comprenda el concepto de árbol de sintaxis, para su uso en proyectos futuros.

Índice

Descripción de la solución

El segundo proyecto de laboratorio consiste en el desarrollo un *framework* que permitirá la interpretación de un lenguaje de programación orientado a objetos, de tipo de datos estático y fuertemente tipado.

Flujo de la aplicación

El proyecto se desarrollará para el lenguaje de programación ***pyUsac***. Este lenguaje consistirá en uno o múltiples de archivos de tipo ***pyUsac***. El flujo de ejecución será marcado por el primer archivo encontrado que contenga un método estático *main*.

Primero se ejecutan todos imports (si hubieran), luego se capturan los símbolos dentro del archivo (clases, métodos y funciones) y luego se ejecutan todas las instrucciones que estén fuera de estos y se ejecuta el método main.

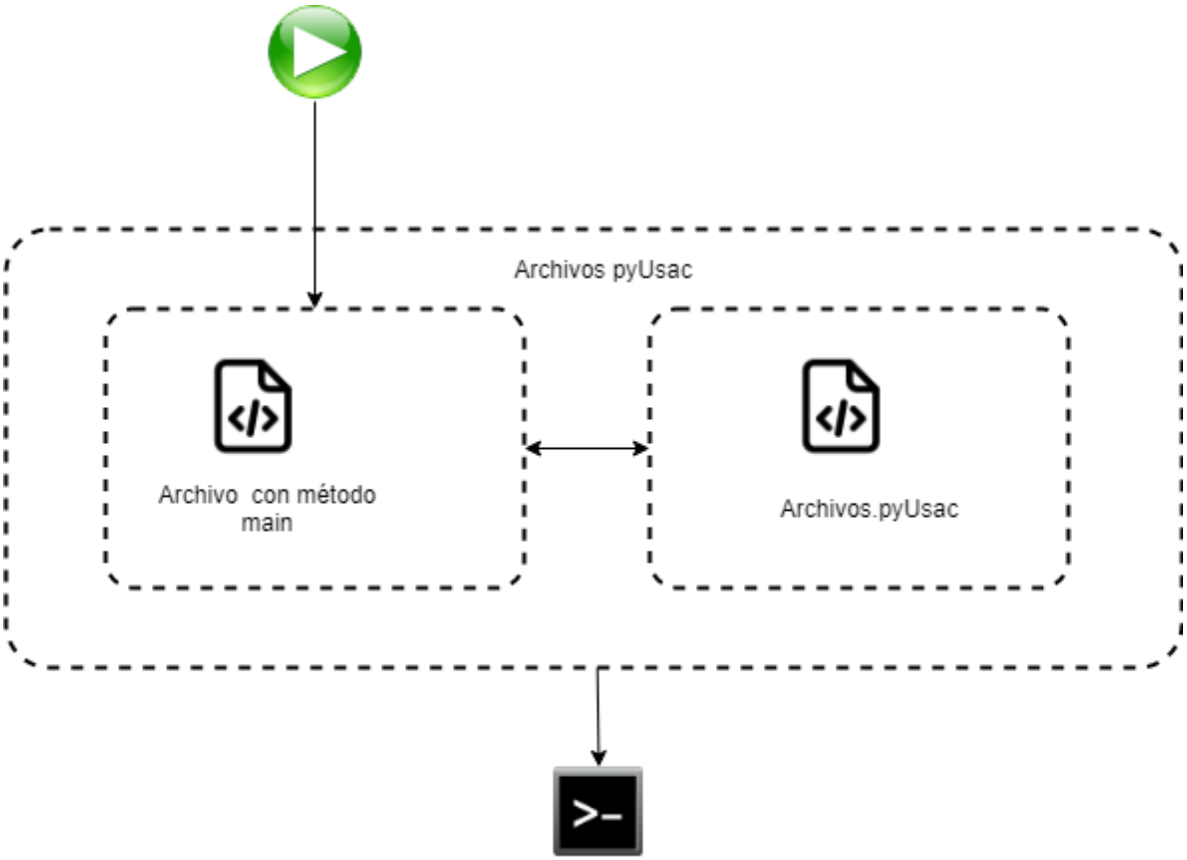

ILUSTRACIÓN 1 FLUJO DE EJECUCIÓN DE UN PROYECTO PYUSAC

Interfaz gráfica

Se contará con una interfaz gráfica que permitirá poder realizar la edición de los diferentes archivos PyUsac. Se deberá desarrollar una aplicación que cuente con una interfaz gráfica capaz de proveer al desarrollador las herramientas para poder desarrollar sus proyectos.

Se deberá de contar con las siguientes secciones:

1. **Área de menú**
    a. *Crear Archivo*
    b. *Guardar Archivo*
    c. *Guardar como*
    d. *Cerrar Archivo*
2. **Área de edición:** La herramienta debe permitir editar varios archivos a la vez.
3. **Área de reportes**
    a. *Área de errores:* Debe mostrarse tabulados los errores encontrados durante la ejecución.
    b. *Área de Consola:* Consola de salida para los programas ejecutados.

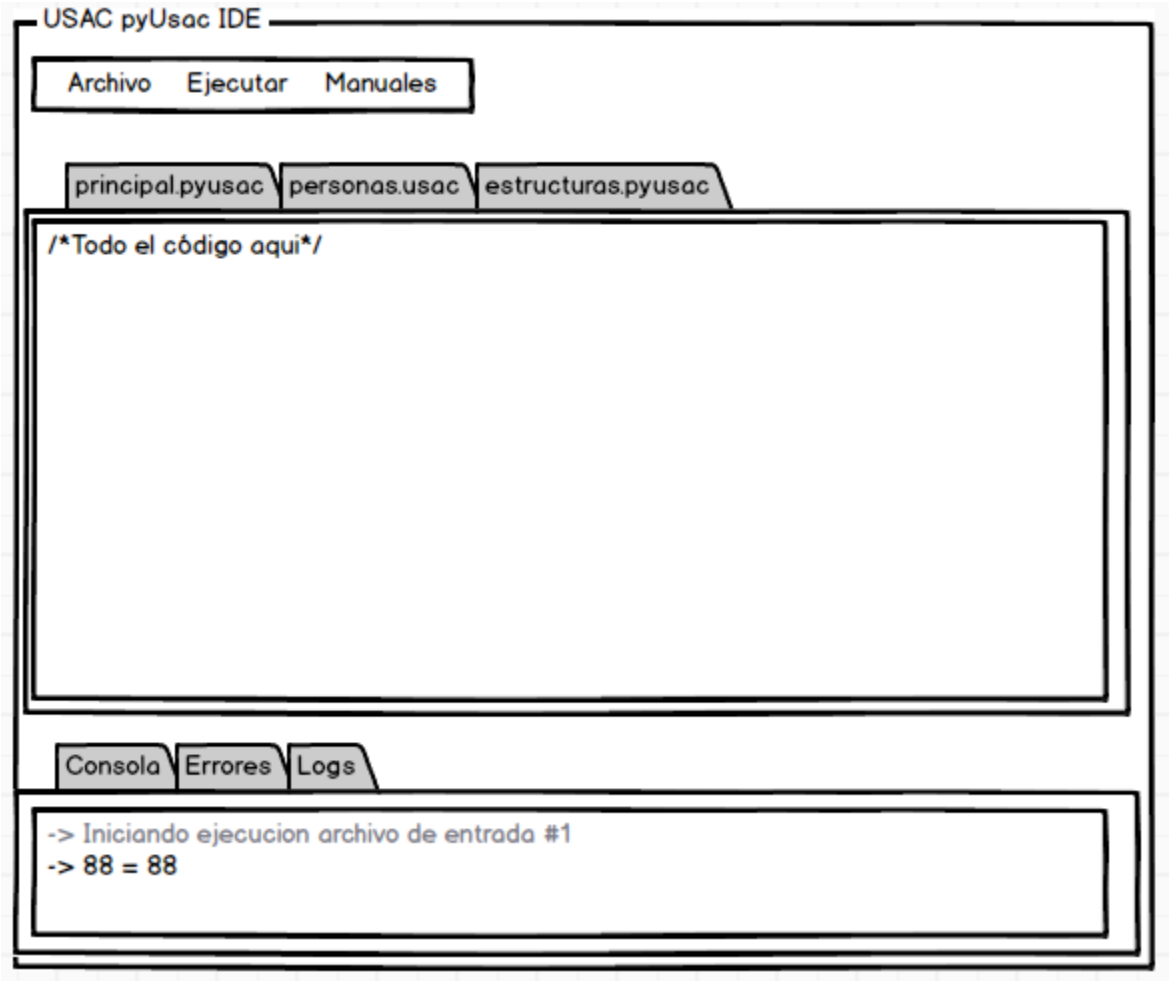

ILUSTRACIÓN 2. PROPUESTA DE INTERFAZ GRÁFICA

Descripción del lenguaje

Intérprete de varias pasadas

El lenguaje de programación **pyUsac** necesitará de un intérprete de dos pasadas. En la primera pasada se almacenarán, en un entorno, los símbolos que se especifiquen en el archivo de entrada. Los símbolos a almacenar serán los siguientes:

- Métodos
- Funciones
- Clases

Luego, en la segunda pasada, se ejecutarán las distintas instrucciones que se encuentren fuera de los métodos, clases y funciones y luego de esto se ejecuta el método main. Es decir, el intérprete a desarrollar será de tipo ***Top-Down*** para su segunda pasada.

Archivos PyUSAC

Los archivos **pyUsac** tendrán la extensión [.pyUsac] y constarán de lo siguiente:

- Lista de importaciones (pueden o no venir)
- Lista de clases, funciones, métodos, instrucciones y un método main.

Case Insensitive

El lenguaje de programación **pyUsac**, será un lenguaje insensensitve case, es decir que no hará distinción entre mayúsculas y minúsculas en palabras reservadas e identificadores.

- Por ejemplo, la palabra reservada true será lo mismo que TRUE, tRue, etc.
- En cuanto a los identificadores, por ejemplo, el identificador contador y el identificador CONTADOR o ConTador serán equivalentes.

Comentarios

El lenguaje de programación **pyUsac** permitirá comentarios de una sola línea o bien de varias líneas.

- En cuanto a los comentarios de una sola línea comenzará con la palabra reservada “//” consistente en dos barras juntas y termina con el carácter de salto de línea “\n”.

```
//comentario de una sola línea
```

- En cuanto a los comentarios de varias líneas comenzarán con la palabra reservada “/*” y terminará con la palabra reservada “*/”.

```
/* comentario de varias líneas
Segunda línea
Tercera línea…. */
```

Tipos de datos

Definición de tipos de datos primitivos

El lenguaje de programación pyUsac tendrá los siguientes tipos de datos primitivos:

Tipo	Descripción	Ejemplo
Char	Un único carácter encerrado entre comillas simples.	'a' 'x'
String	Conjunto de caracteres encerrados entre comillas dobles.	"a" "compi 1"
Int	Valor de numérico sin decimales con tamaño de 4 bits, es decir con un mínimo de -2147483647 y un máximo de 2147483648.	33 66 1693
Double	Valor de tipo numérico con decimales con un tamaño de 32 bits.	3.1416 666.661 777.122340
Boolean	Valor que representa un valor lógico. Se usará la palabra reservada true y false respectivamente.	true false
Null	Valor que representa la ausencia de valor.	null

TABLA 1 TIPOS DE DATOS PRIMITIVOS DE PYUSAC

Nota:

Al momento de obtener un valor y su respectivo tipo, habrá que verificar el valor no sobrepase el valor máximo o valor mínimo. En caso de que esto sucediera, se deberá de reportar un error de semántica.

Tipos de datos compuestos (Tipos de datos de referencia)

El lenguaje de programación **pyUsac** permite la creación y utilización de tipos de datos compuestos que permite empaquetar diferentes tipos de datos. Los tipos de dato compuestos que se podrán utilizar en **pyUsac** son los siguientes:

- Clases
- Arreglos

Nota:

Estos tipos de datos serán explicados en secciones posteriores.

Expresiones

Se denomina expresión a toda instrucción que al interpretarse devuelve tanto un valor explícito como un tipo.

En la siguiente sección se describirán cada una de las expresiones soportadas por el lenguaje de programación pyUsac las cuales se clasifican en tres grandes grupos:

- *Expresiones Literales*: estas se refieren a los valores de expresiones de datos de tipo primitivo.
- *Expresiones aritméticas:* Estas expresiones se refiere a todas las operaciones aritméticas de tipo binaria (operaciones de dos operandos). Estas pueden devolver diferentes tipos de valores según la operación y el tipo de dato de los operandos.
- *Expresiones relacionales:* Estas expresiones se refiere a todas las operaciones de comparación entre valores. Su único posible resultado puede ser un valor primitivo de tipo **booleano**.
- *Expresiones lógicas:* Estas expresiones se refieren al tipo de operaciones lógicas que obedecen al álgebra booleana. Su único posible valor resultante es un valor de tipo primitivo **booleano**.

Nota: En cualquier operación que de como resultado un número con más de 5 decimales, el valor se truncará hasta el quinto decimal.

Signos de agrupación

El lenguaje pyUSac tendrá soporte para signos de agrupación que indican la precedencia de las operaciones. Para agrupar expresiones se utilizarán los paréntesis para indicar el inicio "(" y el final ")" de la agrupación.

Ejemplo:

var suma **=** (a +b)/c;

Expresiones Aritméticas

Las expresiones soportadas por el lenguaje de programación pyUsac son las siguientes:

- Adición (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Potencia (pow)

Adición

La operación será indica con el símbolo “+” y seguirá el siguiente sistema de tipos:

Operandos	Resultado	Ejemplo
Toda operación con al menos un operando de tipo primitivo String	String	“hola “ + “mundo” = “hola mundo” 66.66 + “ = 66.66” = “66.66 = 6.66” ‘a’ + “becedario” = “abecedario true + “hola” = “truehola” “::=” + false = “::=false”
Double + Double Double + int Double + char int + Double char + Double	Double	10.0 + 10.33 = 20.33 10.66 + 3 = 13.66 10.66 + ‘a’ = 107.66 10 + 6.66 = 16.66 ‘a’ + 1.0 = 98.0
Int + int int + char char + int	Int	10 +33 = 33 10 + ‘b’ = 108 ‘c’ + 1 = 100

TABLA 2. RESULTADOS DE OPERACIÓN SUMA

Nota:

- El valor de un valor primitivo de tipo char será el valor del carácter como una cadena cuando el resultado sea de tipo String y el valor será el valor entero ASCII cuando se trate de una operación que dé un valor númerico como resultado.
- Toda combinación que no se encuentre en el sistema de tipos anterior, será tomada como inválida y deberá de reportarse como un error semántico.

Resta

La operación será indica con el símbolo “-” y seguirá el siguiente sistema de tipos:

Operandos	Resultado	Ejemplo
Double - Double Double - int Double - char int - Double char - Double	Double	10.0 - 10.33 = -20.33 -10.66 - 3 = -13.66 10.66 - ‘a’ = -87.66 10 - 6.66 = 3.44 ‘a’ - 1.0 = 96.0
Int - int int - char char - int	Int	10 - 33 = 23 10 - ‘b’ = -88 ‘c’ - 1 = 98

TABLA 3. RESULTADOS DE OPERACIÓN RESTA

Nota:

- Toda combinación que no se encuentre en el sistema de tipos anterior, será tomada como inválida y deberá de reportarse como un error semántico.
- El valor a tomar cuando se opere con un valor de tipo primitivo char será su valor ASCII.

Multiplicación

La operación será indica con el símbolo "*" y seguirá el siguiente sistema de tipos:

Operandos	Resultado	Ejemplo
Double * Double Double * int Double * char int * Double char * Double	Double	10.0 * 3.3 = 33.0 -10.05 * 3 = -30.15 10.0 * 'a' = 970.0 10 * 6.66 = 66.6 'a' * 1.0 = 97.0
Int * int int * char char * int	Int	10 * 33 = 333 10 * 'b' = 108 'c' * 1 = 100

TABLA 4. RESULTADOS DE OPERACIÓN MULTIPLICACIÓN

Nota:

- Toda combinación que no se encuentre en el sistema de tipos anterior, será tomada como inválida y deberá de reportarse como un error semántico.
- El valor a tomar cuando se opere con un valor de tipo primitivo char será su valor ASCII.

División

La operación será indica con el símbolo "/" y seguirá el siguiente sistema de tipos:

Operandos	Resultado	Ejemplo
Double / Double Double / int Double / char int / Double char / Double	Double	10.0 / 3.3 = 3.00300 -10.05 / 3 = -3.35 10.0 / 'a' = 0.10309 10 / 6.66 = 1.50150 'a' / 1.0 = 97.0
Int / int int / char char / int	Int	10 / 33 = 0 10 / 'b' = 0 'c' / 1 = 99

TABLA 5. RESULTADOS DE OPERACIÓN DIVISIÓN

Nota:

- Toda combinación que no se encuentre en el sistema de tipos anterior, será tomada como inválida y deberá de reportarse como un error semántico.
- El valor a tomar cuando se opere con un valor de tipo primitivo char será su valor ASCII.

Potencia

La operación será indica con el símbolo "/" y seguirá el siguiente sistema de tipos:

Operandos	Resultado	Ejemplo
Double pow Double Double pow int Double pow char int pow Double char pow Double Int pow int int pow char char pow int	Double	10.0 pow 3.3 = 1995.26231 -10.05 pow 3 = -1157.625 10 pow 6.66 = 1.50150 'a' pow 1.0 = 97.0

TABLA 6. RESULTADOS DE OPERACIÓN POTENCIA

Nota:

- Toda combinación que no se encuentre en el sistema de tipos anterior, será tomada como inválida y deberá de reportarse como un error semántico.
- El valor a tomar cuando se opere con un valor de tipo primitivo char será su valor ASCII.
- Ya que la operación potencia es una de las que puede aumentar de manera grande, verificar siempre si el valor no es mayor al permitido.

Expresiones Relacionales

Las expresiones soportadas por el lenguaje de programación pyUsac son las siguientes:

- Mayor (>)
- Menor (<)
- Igual (==)
- Desigual (<>)
- Mayor o igual que (>=)
- Menor o igual que (<=)

Operandos	Operador	Ejemplo
Int [Operador] int Int [operador] double double [operador] int double [operador] double	<, > , <= , >=	10 > 10 = false 10 > 10.5 = false 10.0 > 30 = false 10.0 >= 10.4 = false

int [operador] char char [operador] int char [operador] double double [operador] char char [operador] char		97 <= 'a' = true 'a' < 97 = false
Todas las combinaciones anteriores String [operador] String	== , <>	10 == 10 = true 10.0 == 10 = true "USAC" == "USAC" = true "usac" <> "USAC" = true

TABLA 7. EXPRESIONES RELACIONALES

Nota:

- Cualquier combinación que no esté dentro del sistema de tipos anterior será inválida y deberá ser reportado como error de semántica.
- Toda operación relacional devolverá un valor de tipo booleano, es decir true o false.

Expresiones Lógica

Las expresiones soportadas por el lenguaje de programación pyUsac son las siguientes:

- And (&&)
- Or (||)
- Xor(^)
- Not (!) Esta operación, a diferencia de las anteriores es una operación Unaria.

Las operaciones booleanas, seguirán la siguiente tabla de verdad (que es la habitual del álgebra booleana):

Operando No. 1	Operando No. 2	And	Or	Xor	Not
True	**True**	**True**	**True**	**False**	**False**
True	**False**	**False**	**True**	**True**	**False**
False	**True**	**False**	**True**	**True**	**True**
False	**False**	**False**	**False**	**False**	**True**

TABLA 7. EXPRESIONES LÓGICAS

Aumento

Operación aritmética que consiste en añadir una unidad a un dato numérico. El aumento es una operación de un solo operando. El aumento sólo podrá venir del lado derecho de un dato. El operador del aumento es el doble signo más ++.
Especificaciones sobre el aumento:

- Al aumentar un tipo de dato numérico (entero, doble) el resultado será numérico.

- No es posible aumentar el tipo de dato cadena.
- No es posible aumentar tipos de datos lógicos (booleanos).
- El aumento podrá realizarse sobre números o sobre identificadores de tipo numérico.

Operando	Tipo de dato Resultante	Ejemplo
double++	Double	4.56++ = 5.56
int++ char++	int	15++ = 16 'a'++ = 98 = 'b'

Decremento

Operación aritmética que consiste en quitar una unidad a un dato numérico. El decremento es una operación de un solo operando. El decremento sólo podrá venir del lado derecho de un dato. El operador del decremento es el doble signo menos **-**

Especificaciones sobre el decremento:

- Al decrementar un tipo de dato numérico (entero, doble, carácter) el resultado será numérico.
- No es posible decrementar tipos de datos cadena.
- No es posible decrementar tipos de datos lógicos (booleano).
- El decremento podrá realizarse sobre números o sobre identificadores de tipo numérico.

Operandos	Tipo de dato resultante	Ejemplos
double--	double	4.56-- = 3.56
int-- char--	int	15-- = 14 'b'-- = 97 = 'a'

Declaración de variables

Para declarar variables en el lenguaje de programación pyUsac se seguirá la siguiente sintaxis:

Ejemplo:

```
Var a, e, i, o, u = 35.6; // Todas las variables tendrán el valor de 35.6
Var contador = 10;
Var bandera; // Su valor será nulo
```

Consideraciones:

- El tipo de la variable será dado según el valor de la expresión.
- En el segundo caso, si no se le asigna un valor a la lista de variables, su valor por defecto será **NULL**.
- Se debe incluir la palabra reservada **var** antes de la lista de identificadores.

Asignación de variables

El lenguaje de programación pyUsac permite la asignación de valor a una variable, esto se hace indicando el identificador, el signo de asignación será “=”.

Ejemplo:

```
id = "hola mundo";
bandera = getNumero() + 1;
```

Consideraciones:

- Se debe de verificar que la variable exista en el entorno actual, de no existir se deberá de indicar un error de semántica.
- El tipo de la variable cambiará al tipo del nuevo valor asignado.

Funciones nativas

El lenguaje de programación **pyUsac** tendrá las siguientes funciones nativas:

Función nativa Log

La función nativa log tiene como función de imprimir en la consola de la aplicación el valor de la expresión que recibe como parámetro. El valor de la expresión **puede ser de cualquier tipo.**

Ejemplo:

```
log("hola " + "mundo " + 7) ;
```

Función nativa Alert

La función nativa Alert tiene como función generar un mensaje emergente con el valor de la expresión enviada. El valor de la expresión. La expresión que se envía como parámetro puede ser de cualquier tipo.

Ejemplo:

```
alert("Salgo como mensaje " + "emergente " + 4 + 5) ;
```

Función nativa Graph

La función nativa Graph tiene como función el generar, a través del programa Graphviz, la gráfica de la cadena que recibe como parámetro número 2 y guarda la imagen con el nombre que se le indica en el parámetro 1.

Ejemplo:

```
graph ("imagen.jpg", "digraph D { A -> {B, C, D} -> {F} }");

Var nombre = "grafica1";
Var extension = ".jpg";
Var contenido = getCadena(lista1);
graph(nombre + extension, contenido);
```

Nota:

- El primer parámetro indica el nombre con la que se almacenará la imagen generada.
- El segundo parámetro indica el contenido de la cadena que se enviará a graphviz para la generación de la imagen.
- Ambos parámetros en sí son una expresión que debe devolver un valor de tipo String. En caso contrario se deberá indicar error semántico.

Estructuras de control

Una estructura de control por lo general es bloque de instrucciones asociados a cierta prueba como punto único de entrada y según su naturaleza se clasifican (en el caso del lenguaje de programación pyUsac) en sentencias de selección y en iterativas o cíclicas.

Estructuras de control de selección

Una sentencia de control es una sentencia que puede definir el flujo de la aplicación, esto a través de realizar ciertas pruebas a través de expresiones lógicas.

Sentencia de selección IF

La sentencia de control ***if*** consta de un bloque de instrucciones que serán ejecutados si la expresión que se utiliza como prueba da como resultado el valor lógico **true**. En caso de que el resultado de la prueba sea **false**, de existir, se deberán ejecutar la siguiente condición ***else*** o ***else if*** anidada.

Ejemplo:

```
If ( var1 == "Esto") {
        Log("Si");
}else if ( var1 <> "Aquello" ){
        Log ("Casi que si");
}else{
        Log("No ");
}
```

Nota

- Debe verificarse que el valor de la expresión a evaluar dé como resultado un valor de tipo **boolean**, en caso contrario se deberá reportar el error semántico.

Sentencia de selección Switch case

La sentencia de control switch case es una sentencia que permite agilizar la toma decisión múltiple. La sentencia de control switch case consta de un valor de entrada y una lista de posibles casos de posibles valores. Se debe de comprobar si el valor de la variable de entrada es igual al valor esperado por el caso, en caso de ser iguales, se ejecuta el bloque de instrucciones asociadas al bloque. Para indicar el fin de las pruebas se utilizará la palabra reservada 'break'. En caso de no encontrarse un break, al finalizar la ejecución del bloque de instrucciones asociados a un caso, se continuará ejecutando la comprobación de casos.

Existe un caso especial que no tiene un valor para realizar la prueba de entrada, es el caso default, el cual se ejecutará en caso de que todas las pruebas de todos los casos hayan fallado. Este caso siempre se encontrará al final de todos los casos.

Ejemplo:

```
switch(caracter) {
        case 'c':
                log("El carácter es 3");
                var1 = 3;
                break;
        case 'e':
                var1 = 5;
                break;
        default:
                var1 = -1;
        }
```

Estructuras control iterativas

Estas sentencias de control inician o repiten un conjunto de instrucciones si se cumple o mientras se cumpla una condición.

Consideraciones:

- Se debe de verificar que el valor de la expresión de entrada sea de tipo booleano. En caso contrario se abortará la ejecución del programa y deberá de reportarse el error semántico.

Sentencia de control While

Esta sentencia de selección ejecuta un bloque de instrucciones mientras la condición de entrada sea verdadera.

Ejemplo:

```
While( var1 && var2) {
        Alert("Soy la alerta!");
}
```

Sentencia de control DoWhile

Esta sentencia de selección ejecuta un bloque de instrucciones una vez y luego ejecuta ese bloque de instrucciones si la condición de entrada es verdadera.

Ejemplo:

```
Do{
        Alert("Soy la alerta!");
} While( (a<=20) && flag );
```

Sentencia de control For

Este ciclo ejecutará el número de veces necesarias hasta que la condición se cumpla. Esta tendrá un área de asignación o declaración, una condición y actualización. En la actualización se permiten tanto decrementos como aumentos.

Ejemplo:

```
For(var i=0; i< 10; i++){
        Log("Repetir.....");
}
```

Donde:

- Inicialización es donde se realiza una declaración o una asignación inicial de la variable que va a servir para realizar la verificación de condición. Esta variable es llamada variable de control.
- Condición: Es una expresión que se debe de evaluar para poder determinar la continuidad de las iteraciones.
- Actualización: Es una instrucción que permite la actualización de la variable control. Esta instrucción se ejecuta luego de ejecutarse el bloque de instrucciones asociado.

Sentencias de manipulación de flujo

En el lenguaje de programación pyUsac existen sentencias que permiten controlar el flujo de ejecución de las sentencias selección.

Sentencia Break

Esta sentencia indica el fin de la sentencia de control en la cual se encuentra. Tendrá el siguiente comportamiento:

- En caso de la sentencia **Switch-case**, al encontrarse esta instrucción, se dará por terminado la comprobación de cada una de las condiciones siguientes al caso actual.

- En caso de encontrarse dentro del bloque de una sentencia de control iterativa, esta indicará que se de dar por termina la ejecución de dicha sentencia de selección. No se siguen ejecutando las instrucciones siguientes a la sentencia Break.

Ejemplo:

```
For(var i=0; i< 10; i++){
        Log("Repetir.....");
        If(i == 2){
                Break;
        }
}
```

Sentencia Continue

Esta sentencia indica el fin de la iteración actual dentro de una sentencia de control iterativa. Es decir, no se continuarán ejecutando las sentencias siguientes y se comienza de nuevo con la siguiente iteración.

Ejemplo:

```
For(var i=0; i< 10; i++){
        Log("Repetir.....");
        If(i == 2){
                continue;
        }
}
```

Sentencia Importar

Esta sentencia servirá para incluir la funcionalidad de otro archivo al copiar todas las sentencias contenidas en el mismo, funciona como un include de C. Tomará como parámetro un string que contenga el nombre del archivo a incluir. Si solo se coloca el nombre del archivo, se asumirá la misma ruta del archivo donde está declarada la sentencia incluir.

Ejemplo:

```
importar("archivo1.pyusac");
importar("carpeta/archivo2.pyusac");
```

Consideraciones:

- Se debe de verificar que el archivo indicado por la ruta, exista, en caso contrario, inidicar error de tipo semántico.

Métodos

Un método es un bloque de instrucciones encapsuladas bajo un id que pueden o no recibir parámetros para su ejecución. Al terminar su ejecución, no devuelve ningún valor.

Ejemplo:

```
function void sumar(var entero1, var entero2){
        var suma = entero1+entero2;
        log(suma);
}
```

Donde:

- La lista de parámetros formales es una lista separada por coma (‘,’) de variables que constan de un nombre. Estas variables existirán en un nuevo entorno, que se crea al realizar una llamada al método y servirá como interfaz entre el entorno actual, y el nuevo entorno que se produce con la llamada al método.

Funciones

Una función con retorno es un bloque de instrucciones encapsuladas bajo un id que **pueden o no** recibir parámetros para su ejecución. Al terminar su ejecución, este deberá de retornar un valor del tipo del cual se haya declarado utilizando la sentencia RETURN.

La función puede retornar cualquier tipo de dato, tanto primitivo como compuesto tales como objetos y arreglos.

Ejemplo:

```
function sumar(var entero1, var entero2){
        return entero1+entero2;
}
```

```
function generarArrelgo(var size)
{
        Array arreglo[size+1];
        Log(“Se ha creado un arreglo con valores nulos de tamaño “ + (size +1));
        return arreglo;
}
```

```
function generarHijo(var nombre, var edad)
{
        If(edad>=18)
          {
                  Return new Persona(nombre, edad);
          }
          Return null;
}
```

Ejemplo:

Consideraciones:

- Se debe de verificar de forma semántica que una función siempre devuelva un valor a través de la sentencia RETURN.
- Los parámetros de tipo primitivos serán enviados por VALOR y los objetos por REFERENCIA.
- PyUsac permite la recursividad, es decir un método puede llamarse a sí mismo. En ese caso, se creará un nuevo entorno para dicha nueva llamada.

Declaración de arreglos

Se pueden realizar declaraciones de arreglos dentro del lenguaje de la aplicación, por lo que la manera de declaración y asignación es parecida a la de variables, pero a comparación de una variable normal, un arreglo se declara indicando una lista de dimensiones y el tamaño de cada dimensión.
Ejemplo:

```
Var arr1 [a+b+(i+(5+1)*id1)] ;          //con 1 dimensión, inicializado en null
Var arr3, arr4 [contador][5*num+5-4];    //con 2 dimensiones, inicializado en null
```

Consideraciones:

- Las dimensiones serán limitadas a un máximo de 3 dimensiones, esto con el fin de poder facilitar la programación.

- También permitirá asignar valores al arreglo cuando se declare, si no se asigna un valor todas las casillas comienzan con valor nulo o que no tienen valor asignado. Para poder asignar al momento de inicializar se utilizará las llaves "{" "}" para indicar los arreglos para cada dimensión y se separa con coma por cada posición o casilla del arreglo.

```
Una dimensión: {<expresion>,<expresion>, ……,<expresion>}
Dos dimensiones: { {<expresion>,…,<expresion>}, {<expresion>,…,<expresion>}}
Tres dimensiones:
{{{<expresion>,…,<expresion>},…,{<expresion>,…,<expresion>}},{{<expresion>,…,<expresion>},…,{<expresion>,…,<expresion>}}}
```

Ejemplo:

```
Var arr0 [1+2] = {5, 10, 15} ; //En este caso arr0 tiene en la primera posición 5, 10 en la segunda y 15 en la tercera. Es un arreglo con 3 posiciones.

Var arr10, arr20, arr30 [2] [1+1+1] = {
                                        {id1, id2/2, 15*23},
                                        {20, var1^2, 30}
                                       };
Var arr10, arr20, arr30 [2][3][4] = {
                                        {
                                           {5+2, 4, var3, 2},
                                           {8, 3, var15, 9},
                                           {10, var4, var3, 11}
                                        },
                                        {
                                           {20, var1, 30, 23},
                                           {35, 42, 10, 3},
                                           {89, 6, 34, 52}
                                        }
                                    };
```

Uso de arreglos

Una vez definido el arreglo se puede acceder al valor de cada posición del arreglo.

Ejemplo:

```
VarEnt1 = arr0[2+3][3] * 3; // acceso al arreglo 0 en la posición 5 y 3
Var2 = arr5[i *1]+5*8+b; // acceso al arreglo5 en la posición i * 1
```

Reasignación de las posiciones de los arreglos

Una vez definido el arreglo se puede volver a definir el valor de cada posición del arreglo.

Ejemplo:

```
arr0[1*2][4] = arr0[1][0] * 7; // reasignación al arreglo 0 en la posición 2 y 4
arr5[3+x] = 5*8+b; // reasignación al arreglo5 en la posición 3+x
```

Nota:

Si el resultado de la expresión excede el tamaño de la dimensión, deberá mostrarse como un error semántico.

Declaración de clases

Clases

Una clase es un conjunto de variables, funciones o métodos. Las especificaciones para las clases son las siguientes:

- Las listas de variables, funciones y métodos son opcionales.
- Un archivo de entrada puede tener varias clases declaradas.
- Se podrá crear objetos con todas las clases declaradas dentro del archivo de entrada.
- Las variables globales pueden ser de tipo clase.

Ejemplo:

```
Class objeto {
        var var1 = 3;
        function sumar(var sumando1, var sumando2){
                return sumando1 + sumando2;
        }
}
```

Declaración de instancias de clase(Objetos)

Para declarar un nuevo objeto se usará la palabra reservada **new** con la que se creará una instancia del objeto indicado, con sus variables globales y métodos implementados. Tendrá la siguiente sintaxis.

Ejemplo:

```
Var c1 = new Clase1();
Var c2 = null;
Var c2 = new Clase2();
c2.numero = 3+4;
log(c2.numero );
```

Acceso a variables, funciones y métodos de una clase

Para acceder a una variable se usará el nombre de la instancia de la clase, el símbolo "." y después el nombre de la variable, función o método que se quiere acceder.

Ejemplo:

```
var numero1 =  c1.num1 ;
var num2 =  c1.funcion1(2*var1, 4.5*var3, 45) ;
var c2 = c1.funcionRetornarClase() ;
var cad1 = c2. concatenar("hola", "mundo") ;
var arreglo1[2] = funcionRetornarArreglo() ;
C2.metodoPrintEnConsola() ;
```

Reasignación de variables globales del objeto

Se puede reasignar o asignar las variables de una clase previamente declarada.

Ejemplo:

```
c2.variableentera1 = 34 ;
c2.VarCadena = "holamundo" ;
```

Llamada de funciones

La sentencia para poder invocar una función consiste en escribir el nombre de la función y entre paréntesis, la lista expresiones que recibe como parámetros la función, separados con comas. Los valores que puede recibir **como parámetros** son **tipos primitivos, OBJETO, ARREGLOS**. Cumpliendo la siguiente sintaxis:

Ejemplo:

```
IniciarMensajes() ;
C2.funcionSuma(2, var2+5*4) ;
Var arreglo1 [2][3] ;
funcion1(arreglo1) ;
funcionObjeto(new Objeto()) ;
```

Sentencia retornar

Esta sentencia se encargará de retornar el valor de la expresión indicada. Esta sentencia podrá estar en **cualquier segmento** del bloque de una función. Al ejecutarse sentencia, se hará el cálculo de la expresión indicada y **ya no se ejecutarán las sentencias siguientes** (cuando haya sentencias después de esta).

Ejemplo:

```
Return 10 + id ;
Return count() ;
Return flag && flag2 ;
```

Método MAIN

El método main, será el encargado de llevar el control del flujo de la aplicación luego de la ejecución de todas las instrucciones que se encuentre fuera de los métodos, clases y funciones (Se explicó en la sección de Descripción del lenguaje). Tendrá la siguiente sintaxis.

Ejemplo:

```
main(){
        /*Todo el código aquí. */

}
```

Restricciones

- La aplicación deberá desarrollarse en lenguaje C#.
- La herramienta para el análisis léxico y sintáctico deberá usar Irony.
- El proyecto se realizará de manera individual si se detecta algún tipo de copia el laboratorio quedará anulado.
- El estudiante debe de ser capaz de correr su aplicación desde el archivo ejecutable, no se permitirá correr la aplicación desde el IDE.

Nota:

Se recomienda utilizar el árbol de Irony para generar su propio AST y no ejecutar directamente con el que genera Irony.

Entregables

- Aplicación con TODAS las funcionalidades antes descritas en el enunciado.
- Código fuente.
- Gramática escrita en C# utilizando Irony.

Fecha límite de Entrega: domingo 17 de noviembre de 2019 hasta las 23:59 PM.