PROCESADORES DE LENGUAJES

Entrega 1: Definición del lenguaje

Curso 2020/21

---

12 de marzo de 2021

4º Grado de Ingeniería Informática

José María Amusquívar Poppe, Prashant Jeswani Tejwani

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería en Informática

Índice

Introducción

En esta primera fase de la práctica se solicita realizar la especificación del lenguaje que se irá implementando a lo largo del curso. Este lenguaje ha de disponer como mínimo declaraciones de variables numéricas, instrucciones de asignación de expresiones, estructuras de control, rutinas recursivas, así como operaciones de salida para variables y ristras.

Por lo tanto, el lenguaje que se ha decidido implementar se denominará *GTau*, y tendrá una extensión de ficheros *.gta*. La estructura del código estará formada por módulos denominados *function*, y de una función raíz denominada *main,* que será desde la que se ejecuten las demás rutinas deseadas.

```
function example() {
    instrucción1;
    instrucción2;
    ...
}

function int example1() {
    instrucción1;
    instrucción2;
    ...
    int aux = 5;
    return aux;
}

main() {
    example();
    int resultado = example1();
    println(resultado);
}
```

Figura 1. Ejemplo de estructura del código.

Descripción del lenguaje a implementar

A continuación, se describirán las características del lenguaje implementado:

Tipos y declaración de variables

Todas las variables son locales por lo que deberán ser declaradas en cualquier parte de una función y usadas dentro de ella.

El lenguaje a implementar incorporará declaraciones de tipo:

- **Integer:** Utiliza 4 bytes de almacenamiento y los rangos de valores están comprendidos entre -2^{31} y 2^{31}. La declaración se hará de la siguiente manera:

```
int nombreInt = 10;
```

- **Float:** Utiliza 4 bytes y permite representan números en coma flotante. La declaración se hará de la siguiente manera:

```
float nombreFloat = 5.5;
```

- **Double:** Utiliza 8 bytes (con mayor precisión que *float*) y permite representan números en coma flotante. La declaración se hará de la siguiente manera:

```
double nombreDouble = -7.568;
```

- **Boolean:** Almacena los valores verdadero o falso, utiliza 1 byte de almacenamiento. La declaración se hará de la siguiente manera:

```
bool nombreBool = FALSE;
```

- **Char:** Utiliza 1 byte, y se emplea para guardar caracteres de manera individual. La declaración se hará de la siguiente manera:

```
char nombreChar = "C";
```

Tipos y declaración de estructuras de variables

El lenguaje a implementar incorporará declaraciones de estructuras de tipo:

- **String:** Conjunto de caracteres consecutivos de tipo *char*. La declaración se hará de la siguiente manera:

```
string nombreString = "Esto es una prueba";
```

- **Vector:** Estructura mutable que alberga conjunto de elementos consecutivos del mismo tipo. La declaración se hará de la siguiente manera:

```
vect[int] nombreVector = [];
vect[int] nombreVector = [6, 8, 4, 10, -5];
```

- **Diccionario:** Conjunto de pares clave-valor, con claves únicas. La declaración se hará de la siguiente manera:

```
dict[int, string] nombreDict = [];
dict[int, string] nombreDict = [5:"Cinco", 4:"Cuatro"];
```

Estructuras de control

El lenguaje a implementar incorporará estructuras de controle de tipo:

- **IF-ELSIF-ELSE:** Estructura de control más sencilla. Se usa para decidir si una instrucción o bloque de instrucciones se ejecuta dependiendo de una condición. La sintaxis será de la siguiente manera:

```
if (condición1) {
    instrucción1;
    instrucción2;
    ...
} elsif (condición2) {
    instrucción1;
    instrucción2;
    ...
} else {
    instrucción1;
    instrucción2;
    ...
}
```

- **FOR:** Permite ejecutar un bloque de instrucciones un número determinado de veces (según el incremento) mientras se cumpla una condición. La sintaxis será de la siguiente manera:

```
for ((int | float | double) i = inicio; condiciónDeParada; i=i+incremento | i+=incremento) {
    instrucción1;
    instrucción2;
    ...
}
```

- **WHILE:** Permite ejecutar un bloque de instrucciones un número determinado de veces mientras se cumpla una condición. La sintaxis será de la siguiente manera:

```
while (condiciónDeParada) {
    instrucción1;
    instrucción2;
    ...
}
```

- **BREAK:** Permite finalizar una ejecución de un bucle. La sintaxis será de la siguiente manera:

```
while (condiciónDeParada) {
    instrucción1;
    instrucción2;
    if (condición1) {
        break;
    }
}
```

- **GOTO 'etiqueta':** Salto incondicional mediante una etiqueta permite saltar de un punto del programa a otro. Estos saltos deberán ser locales (dentro de una misma función). La

```
#PRUEBAS:
instrucción1;

if (condición1) {
    instrucción1;
    instrucción2;
    ...
} else {
    goto PRUEBAS;
}
```

etiqueta debe estar declarada empleando un **#** al inicio. La sintaxis será de la siguiente manera:

Operadores aritméticos

- **Suma (+):** Realiza la suma entre dos variables numéricas.
- **Resta (-):** Realiza la resta entre dos variables numéricas.
- **Multiplicación (*):** Realiza la multiplicación entre dos variables numéricas.
- **División (/):** Realiza la división entre dos variables numéricas.
- **Potencia (^):** Realiza la potencia de una variable respecto a otra.
- **Módulo (%):** Realiza el módulo entre dos variables numéricas.

Operadores lógicos

- **AND:** Verdadero cuando se cumplen todas las condiciones. La sintaxis será de la siguiente manera:

```
if (condición1 and condición2 and condición3) {
    ...
}
```

- **OR:** Verdadero cuando se cumplen al menos una de las condiciones. La sintaxis será de la siguiente manera:

```
if (condición1 or condición2) {
    ...
}
```

- **NOT:** Niega una condición. La sintaxis será de la siguiente manera:

```
if (not(condición1) {
    ...
}
```

Operadores de asignación

- **Asignación (=):** Asigna el valor a una variable.

Operadores aritméticos de asignación

- **Suma (+=):** Suma y asigna el valor actual de la variable de la izquierda con la variable de la derecha.
- **Resta (-=):** Resta y asigna el valor actual de la variable de la izquierda con la variable de la derecha.
- **Multiplicación (*=):** Multiplica y asigna el valor actual de la variable de la izquierda con la variable de la derecha.
- **División (/=):** Divide y asigna el valor actual de la variable de la izquierda con la variable de la derecha.

Operadores de comparación

- **Igualdad (==):** Compara si el valor de dos variables del mismo tipo es igual.
- **Distinto (!=):** Compara si el valor de dos variables del mismo tipo no es igual.
- **Mayor o igual (>=):** Compara si el valor de la variable de la izquierda es mayor o igual que el de la derecha.
- **Menor o igual (<=):** Compara si el valor de la variable de la izquierda es menor o igual que el de la derecha.
- **Mayor (>):** Compara si el valor de la variable de la izquierda es mayor que el de la derecha.
- **Menor (<):** Compara si el valor de la variable de la izquierda es menor que el de la derecha.

Documentación

El lenguaje dispondrá de una utilidad para añadir comentarios en medio del código, qué podrán ser de una o más líneas. La sintaxis será de la siguiente manera:

```
/* Esto es un comentario de una línea */
/* Esto es un comentario de varias
function string reading() {
    instrucción1;
    return "";
}
líneas de código */
```

Funciones nativas

Imprimir por pantalla (print('variable1')): Esta función se encarga de mostrar por consola el contenido de la variable pasada como parámetro. Esta variable podrá ser de tipo *INT*, *DOUBLE*, *FLOAT*, *CHAR* o *STRING*. En el caso de las variables numéricas, se realizará un *casteo* automático a variables de tipo *CHAR* o *STRING*.

Imprimir por pantalla con salto de línea (println('variable1')): Mismo funcionamiento que la función *print()*, añadiendo un salto de línea al final.

Casting de variables: Se proporcionará una serie de funciones que permitirán realizar la conversión del tipo de variable a otra requerida.

- **int('variable'):** Recibe como parámetro una variable de tipo *DOUBLE*, *FLOAT*, *CHAR* o *STRING*, y convierte dicho valor en una variable de tipo *INT*. En el caso de las dos primeras, el valor resultante será truncado. En el caso de *CHAR*, si el valor es numérico, según *ASCII*, se devuelve el valor numérico, en otro caso, se devuelve su valor *ASCII*. En el caso de la *STRING*, si la ristra contiene algún valor no numérico, provocará una excepción.

- **double('variable'):** Recibe como parámetro una variable de tipo *INT*, *FLOAT*, *CHAR* o *STRING*, y convierte dicho valor en una variable de tipo *DOUBLE*. En el caso de la primera, devuelve el mismo número con un decimal igual a cero, en el caso de *FLOAT* devuelve el mismo valor. En el caso de *CHAR*, si el valor es numérico, según *ASCII*, se devuelve el valor numérico seguido de un decimal igual a cero, en otro caso, se devuelve su valor *ASCII* seguido de un decimal igual a cero. En el caso de la *STRING*, si la ristra contiene algún valor no numérico, provocará una excepción. Al igual que este casting, también existe otro específico para variables a tipo *FLOAT*, que sigue las mismas reglas anteriores.

- **char('variable'):** Recibe un variable de tipo *INT* o *STRING*. El entero debe ser mayor o igual que cero y menor o igual que 9, ya que si no saltará una excepción. En el caso de *STRING*, su longitud debe ser igual a 1.

- **string('variable'):** Recibe cualquier tipo de variable, y convierte ésta a una de tipo *STRING*.

Operaciones con vectores:

- **size('vector'):** Devuelve el tamaño actual del vector.
- **append('vector', 'elemento'):** Añade el elemento pasado como parámetro a la última posición del vector. Este elemento debe ser del mismo tipo que los elementos del vector, además que no puede ser ninguna estructura.
- **pop('vector', 'elemento'):** Elimina y retorna el último elemento del vector, pudiendo asignarlo a otra variable.
- **clear('vector'):** Elimina todos los elementos del vector.
- **get('vector', 'indice'):** Devuelve el elemento del vector situado en la posición 'indice'.
- **clone('vector'):** Retorna una copia del vector pasado como parámetro;

Operaciones con diccionarios:

- **size('diccionario'):** Devuelve el tamaño actual del diccionario.
- **append('diccionario', 'clave', 'valor'):** Añade la pareja clave-valor pasados como parámetro al diccionario. La clave y el valor deben ser del mismo tipo que los contenidos en el diccionario.
- **pop('diccionario', 'clave'):** Elimina la pareja clave-valor del diccionario cuya clave se corresponde con la pasada como parámetro, y retorna el valor, pudiendo asignarlo a otra variable.
- **clear('diccionario'):** Elimina todos los elementos del diccionario.
- **get('diccionario, 'clave'):** Devuelve el valor del diccionario cuya clave se corresponde con la pasada como parámetro.
- **clone('diccionario'):** Retorna una copia del diccionario pasado como parámetro.

Operaciones con strings:

- **size('string'):** Devuelve el tamaño actual de la *string*.
- **append('string', 'string'|'char'):** Retorna la concatenación de la primera *string* con el segundo parámetro.
- **get('string', 'indice'):** Devuelve el *char* que está situado en la posición señalada por 'indice'.

Declaración de funciones

El lenguaje dispondrá de módulos de código denominados *function*, los cuales se encargan de ejecutar una serie de instrucciones. Estas funciones se clasificarán en dos, las que devuelven un valor, y las que no. Algunos ejemplos de declaración de funciones serían:

```
function example() {
    instrucción1;
    instrucción2;
    ...
}

function int example1() {
    instrucción1;
    instrucción2;
    ...
    int aux = 5;
    return aux;
}
```

En el ejemplo anterior, *int* se corresponde con el tipo de variable que devolverá la función, mientras que *example* y *example1* se corresponde con el nombre de la función.

Función principal del fichero ".gta"

Existirá una función principal que será el punto de partida de la aplicación, esta función se denominará *main*, y tendrá que estar localizada al final del fichero. El fichero debe poseer como mínimo dicha función.

```
main() {
    example();
    int resultado = example1();
    println(resultado);
}
```

Baterías de test

Fichero adjunto *test.gta*

Aplicación demostrativa del lenguaje

Fichero adjunto *uso.gta*