Procesadores de lenguajes

# Trabajo de curso

Creación de un lenguaje y su compilador a código Q. Lenguaje creado: “Atomic Kitty”.

Gabriel Jiménez Perera - 44745858W
Alberto Casado Garfia - 45352480E

# Contenido

# Definición del lenguaje

## Introducción

El lenguaje que hemos pensado está basado en Python y en C, principalmente. Pretendemos la sencillez del C puro con la comodidad de Python, obligando a que el código quede bien organizado mediante el uso de tabuladores. También queremos incorporar nuestra versión de las tuplas de Python, dejando de lado los vectores de C. En cuanto al tipado, será de carácter fuerte.

Tipos primitivos del lenguaje: int, long, float, double, char, string, bool.

Las funciones pueden retornar void y se pueden crear nuevos tipos con tuplas.

Otras palabras reservadas: and, or, if, elif, else, in, not, is

## Notas sobre tipos

Entre distintos tipos no se permiten conversiones implícitas y las conversiones explícitas se realizarán mediante el cast de C, solo que utilizando llaves en lugar de paréntesis *{tipo-destino} variable*.

## Tuplas

Las tuplas son elementos híbridos entre las tuplas de Python y los vectores de C. Pueden tener varios tipos predefinidos en su interior. No se permite modificar sus elementos.

## String

Las ristras de caracteres no tendrán un tamaño máximo.

### *Declaración y asignación*

**(tipo1, tipo2,...) nombre = (expresión que devuelve tipo1,...)**

```
(int, int) tupla = (0,1)

(int, string) tupla2 = (3, "hola")

(string, int, float) tupla3 = ("mundo", 4, 0.5)

(string, (int, float)) tupla4 = (tupla3[0], (4, 0.5))


tupla[0] = 3 #no permitido, las tuplas no pueden modificarse
```

### *Acceso a sus elementos*

Por índice como los vectores de C.

```
(string, int) tupla4 = (tupla2[1], tupla2[0])
```

Si la tupla está compuesta por elementos con el mismo tipo, podrán usarse expresiones que devuelvan un entero para acceder a los elementos. En caso contrario solo se podrán usar números enteros literales.

(int, int) tupla1 = (1,2)

(int, char)tupla2 = (1,'a')

int m

int n1 = tupla1[0]  #Permitido

int n2 = tupla1[m]  #Permitido

int n3 = tupla2[0]  #Permitido

¿? n4 = tupla2[m]  #No permitido

## Operadores con números

Se utilizarán los operadores básicos para número de C: +, -, *, /. También se utilizarán los comparativos <, >, <=, >=. Para comprobar la igualdad se utilizará la palabra reservada "is".

Precedencia de operadores de forma parecida a C. Los que se encuentren en el mismo lugar se interpretarán por orden de izquierda a derecha: en **primer lugar** paréntesis, en **segundo lugar** las multiplicaciones y divisiones, en **tercer lugar**, sumas y restas, en **cuarto lugar**, el operador not, en **quinto lugar** los operadores de comparación, en **sexto lugar** los operadores de igualdad, en **séptimo lugar** el operador and, en **octavo lugar** el operador or y, en **último lugar**, el operador in.

De forma especial, el operador de cast (que utilizan llaves), tendrá la precedencia máxima.

## Declaración de variables

La declaración de variables se realizará de manera muy parecida a Python:

**tipo nombre = expresión**

**nombre = expresión**

int i = 0

i = 1

float n = 2.0

double d = 2.0

long l = 20

string s = "string"

bool b  = true is not false

char c = 'c'

## Asignación de expresiones

**tipo nombre = expresion**

int i = 1+2+(3 + 4)*5/7  (i: 8)

## Expresiones con ristras

Las ristras permitirán ser concatenadas con el operador "+" y se permitirá el acceso a sus elementos mediante índice como las tuplas.

## Funciones del lenguaje

i..j -> Representa un rango de valores similar a [i, j).

i..j,k -> Similar a la función anterior, pero el incremento sería de k en k. i y j pueden ser expresiones numéricas o caracteres, pero k solo puede ser una expresión numérica.

print(string p1) -> Función que imprime por pantalla.

print(string p1, int p2) -> Función que imprime por pantalla, p1 deberá contener "%i" y se sustituirá dicha expresión por p2.

getchar() -> devuelve un carácter escrito por el usuario

exit() -> Función que termina la ejecución del programa.

## Estructuras de control

### Estructuras condicionales

#### *if elif else*

Utilizaremos estructuras condicionales simples muy parecidas a C, pero sin la necesidad de paréntesis para la condición.

```
if condicion1:
  cosas1()
elif condicion2 or condicion3:
  cosas2()
else:
  cosas3()
```

#### *when*

Para la estructura tipo switch, implementaremos nuestra versión del when de Kotlin. Para cada caso solo se permitiría una instrucción y los casos pueden ser negados con el operador not.

```
when expresión:
  x -> instrucción
  …
  else -> instrucción
```

Se ejecutaría la primera instrucción que cumpla x, donde x puede ser:

- Una lista de valores (con al menos un elemento) que se compara con el resultado de la expresión. Si alguno de los valores de la lista es igual *(is)* al resultado de la expresión, se ejecuta la instrucción indicada.

- Una estructura tipo *in* para indicar un rango de valores de tipo *m..n*, por lo que si el *resultado de la expresión >= m and resultado de la expresión < n*, se ejecuta la instrucción correspondiente.

```
when variable:
    0, 1 -> cosas1()
    in 2..10 -> cosas2()
    not in 10..20 -> cosas3()
    else -> cosas4()
```

Estructura equivalente en C:

```
if (variable == 0 || variable == 1){
        cosas1();
} else if (variable >= 2 && variable < 10){
        cosas2();
} else if (!(variable >= 10 && variable < 20)){
        cosas3();
} else {
        cosas4();
}
```

## Estructuras de bucle

### *for*

Bucle for, se utiliza declarando o reutilizando una variable tipo int a la que se le asignará los valores dados por la función de rango especificada, por tanto, si la variable utilizada existe se utilizará modificando su valor y, si no existe, se creará una variable nueva tipo entero que se eliminará al terminar el bucle. Además, podrán añadirse condiciones con un and:

```
for i in i..j and condicion:
  cosas()
```

### *while*

Bucle while tipo C con la notación del lenguaje:

```
while condicion:
    cosa1()
    cosa2()
```

## Funciones

### Definición

Las funciones se definirán como en Python, pero con el tipo como en C:

```
void funcion(int p1, int p2):
  cosas()
```

Como parámetros y como resultado, se pueden pasar y devolver cualquier tipo primitivo del lenguaje. Además, puede no devolverse nada si se indica como tipo de resultado *void*:

```
(float, float) funcion((int, int) p):
  cosas()


void función(int p):
  cosas()
```

El paso de parámetros y la devolución de resultados se realiza por valor para los tipos primitivos y, para las tuplas, se pasa una referencia de forma implícita (sin especificar de alguna manera que es un puntero), ya que las tuplas no son mutables y solo son de lectura.

## Código fuera de las funciones

Al igual que en Python se comenzará a ejecutar el código que no esté dentro de ninguna función. Las variables que no se declaren dentro de una función tendrán ámbito global. Las variables declaradas dentro de una función tendrán un ámbito solo dentro de dicha función. De igual forma, las variables declaradas dentro de un bloque (while, for…) tendrán ámbito solo dentro de dicho bloque.

## Ejemplo de funcionamiento del lenguaje

### Fibonacci

```
(int, int) fibo (int p):
  if p is 0:
    return (0, 0)
  elif p is 1:
    return (1,0)
  else:
    (int, int) tupla = fibo(p-1)
    int i = tupla[0]
    int j = tupla[1]
    return (i + j, i)
```

*Explicación*

fibo(6)->(8,5)

fibo(5)->(5,3)

fibo(4)->(3,2)

fibo(3)->(2,1)

fibo(2)->(1,1)

fibo(1)->(1,0)

---