Compiladores 2023-2
Facultad de Ciencias UNAM
Jelly

El lenguaje que buscamos compilar se llama *Jelly* y en este documento se va a dar su especificación sintáctica además de sus diferentes posibilidades.
Jelly es un lenguaje de programación imperativo procedural. Los programas consistirán de un único archivo fuente que contiene funciones, métodos y una única función `main`, la ejecución del programa consiste de la evaluacipn de esta última.

1. Tipos.

Los tipos que Jelly puede manejar son dos: el tipo entero (`int`) y el tipo booleano (`bool`) cuyos únicos dos miembros son `True` y `False`. También existe el tipo arreglo `T[]` para `T` `bool` o `int`.
Cada uno de de los procedimientos contenidos en el archivo fuente puede o no regresar algo, decimos que un procedimiento que tiene tipo de regreso es un método y uno que no lo tiene es una función.
En principio no vamos a tener tipo `string` pero se puede implementar con relativa facilidad y es uno de los ejercicios del proyecto.

2. Identificadores.

Las variables, funciones y métodos requieren un identificador, este puede estar conformado por letras, números y guiones bajos pero necesariamente debe comenzar con una letra minúscula. La declaración de las variables van junto a su tipo y un opcional valor o expresión inicial:

```
temp1:int = 2
temp_2:int
dummyVar:boolean
i:int = f(x)
```

El uso de una variable no inicializada como el caso de `dummyVar` es indefinido, así que en principio nuestro compilador no detectará el uso de variables no inicializadas. Naturalmente, el valor de una variable puede ser cambiado después de su declaracion usando una asignacion:

```
x = x + 1
b = !b
```

El alcance de una variable depende de si es local o global, para las variables locales se encuentra desde el punto en el que es declarada hasta el final del bloque en el que se declara, mientras que para las variables globales es todo el archivo.
Las variables globales se declaran **al inicio del documento**, además de que las variables globales enteras y booleanas deben ser inicializadas en el momento de su declaración.

3. Procedimientos.

Un archivo que representa un programa válido contiene la función `main` y a continuación una secuencia de funciones o métodos.

La declaración de un procedimiento empieza con su nombre, seguido por sus argumentos, el tipo de regreso y la definición de su comportamiento:

```
gdc (var1:int, var2:int): int {
    while(var1 != 0){
        if (var1 < var2) var2 = var2 - var1
        else var1 = var1 - var2
    }
    return b
}
```

4. Particularidades de los tipos

Enteros

Los enteros soportan las opraciones usuales: `+` , `-` , `*` , `/` y `%` además de poder ser comparados con: `==`, `!=`, `<`, `<=`, `>` y `>=`.

Booleanos

Los miembros del tipo `bool` son capaces de operarse con: `&`, `|` y `!`; e igual que los enteros se pueden comparar con `==` y `!=`.

Arreglos

Un arreglo `T[]` es una secuencia mutable de celdas de tipo `T` con longitud fija. Si `a` es un arreglo e `i` es un entero, entonces la expresion `a[i]` significa el valor del arreglo `a` en su posicion `i`. Para que un número represente un índice válido `i` debe ser no negativo y menor que la longitud del arreglo. La expresion `length(a)` da la longitud del arreglo `a`.
Considera la siguiente función que ordena un arreglo:

```
sort(a: int[]){
    i:int = 0
    n:int = length(a)
    while i < n {
        j:int = i
        while j > 0 {
            if a[j-1] > a[j]{
                swap:int = a[j]
                a[j] = a [j-1]
                a[j-1]= swap
            }
            j = j-1
        }
        i = i+1
    }
}
```

Un arreglo puede ser construido especificando sus elementos entre llaves, similar a la sintáxis de Java y C, estos elementos deben ir separados por comas:

```
a: int[] = {1,2,3,4,5}
```

5. Sintáxis adicional

Jelly tiene los siguientes aditivos dulces a su sintáxis:

- Asignación multiple.

  ```
  int a, b, c
  a = b = c = 0
  ```

  No se permiten expresiones dentro de la asignación múltiple.

- Operador de asignación compuesto.

  ```
  a+=5
  b-=a
  ```

- Auto incremento y decremento.

  ```
  i++
  j--
  ```

- If corto.

  ```
  c : int = a > b ? a : b
  ```

- Los comentarios son iguales que en Haskell.

  ```
  --Esto es un comentario de una linea
  {-Esto es un
    comentario
    *multilinea*-}
  ```