El lenguaje *alike*

Procesadores de lenguajes

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Este documento detalla las características del lenguaje de programación ***alike***, que manejaremos a lo largo de las sesiones de laboratorio del curso 23-24. ***alike*** es un lenguaje estructurado sencillo, muy similar a Ada. Entre sus caracterísiticas destacan las siguientes (véanse los ejemplos de la batería de tests suministrada):

1. Los comentarios empiezan con la marca `--` (doble guión) y finalizan al terminar la línea.

2. Se permite la declaración y uso de variables simples (escalares), tanto globales como locales, de tres tipos: *character*, *boolean* e *integer*. La declaración de variables sigue la sintaxis de Ada:

   ```
   i, j, k: integer;
   c, d, e: character;
   v, f: boolean;
   ```

   La declaración de variables globales debe hacerse al principio del programa, y antes de declarar procedimientos o funciones o cualquier instrucción.

3. Los identificadores estarán compuestos por letras, números y el símbolo "_" (carácter *underscore*), con la restricción de que no puede comenzar por un número.

4. El lenguaje es **case-insensitive**, tanto para las palabras clave como para los identificadores. Es decir, no distingue mayúsculas de minúsculas.

5. Ningún símbolo puede llamarse como las palabras reservadas. Es decir, no se permite declarar una variable o una función de nombre "integer", o "if", por ejemplo.

6. Las constantes de tipo carácter se escriben entre comillas simples. El carácter comilla simple se indica mediante tres comillas simples:

```
c := 'a';
c := '"';
c := ''';
```

7. La condición de las instrucciones de selección (`if`) y de iteración (`while`) sólo puede ser de tipo `boolean`.

8. La instrucción de selección se escribe como sigue, siendo las partes `elsif` y la parte `else` opcional:

```
if <expresión> then
    ...
elsif <expresión> then
    ...
elsif <expresión> then
    ...
...
else
    ...
end if;
```

9. La instrucción de iteración se escribe como:

```
while <expresión> loop
    ...
end loop;
```

   Tanto en el caso del `if` como del `while`, es necesario que haya al menos una instrucción en el bloque, no se permite el bloque vacío. Como en el proceso de desarrollo a veces es interesante dejar un bloque vacío, que será completado en una fase posterior, ***alike*** suministra, al igual que Ada, la instrucción nula, que no tiene ningún efecto en la ejecución del programa.

```
while x>0 loop
    null; --a completar más tarde
end loop;
```

10. El lenguaje distingue *funciones* como abstracción de datos y *procedimientos* como abstracción de instrucciones

11. El lenguaje permite la declaración de procedimientos y funciones anidados.

12. En el caso de un procedimiento o función, sus variables locales se declaran al principio de su bloque, antes de cualquier declaración de procedimiento, función o instrucción. Por otro lado, no se pueden declarar variables locales dentro de un bloque `if` o `while`.

13. El programa principal es el procedimiento que engloba todo el código.

14. Cuando un procedimiento o función no tiene parámetros, debe invocarse sin paréntesis.

15. Hay dos formas de paso de parámetros: por valor y por referencia, con la semántica habitual. Los parámetros por referencia se marcan con la palabra clave `ref`. Si no se marca por referencia, se entiende que es por valor. Un ejemplo sería:

```
-- asumimos elementos = 80
procedure inicializar (colonia: ref array(1..80) of boolean) is
    i: integer;
begin
    i := 1;
    while i <= elementos loop
        colonia(i) := (i mod 20) = 0;
        i := i + 1;
    end loop;
end;
```

16. Se permite la escritura de variables y expresiones simples (no de "arrays"), mediante las instrucciones `put` y `put_line` (esta última añade un salto de línea tras la escritura). Como salida, la operación de escritura mostrará por la salida estándar el valor correspondiente a las expresiones de tipo entero o carácter, y las cadenas `true` o `false` en el caso de booleanos. También se permite la escritura de constantes de tipo cadena. La instrucción `put` requiere al menos un argumento; `put_line` puede no tener argumentos (en cuyo caso, no usará paréntesis).

17. La entrada de datos escalares (no arrays, aunque sí componentes de array) se hace mediante la instrucciones `get`. La instrucción `skip_line` salta caracteres de la entrada hasta que logra leer y saltar un *new line*. Ejemplos serían:

```
    n: integer;
    c,b: character;

    get(n,c,b);
    skip_line;
```

    La instrucción `get` requiere al menos un parámetro en su invocación, mientras que `skip_line` no acepta ningún parámetro.

18. Se permite el uso de cadenas de caracteres constantes, aunque solamente para escritura:

```
    put_line("x: ",x);
```

    Las constantes string no usan secuencias de escape. Cuando dentro de un string se desee usar el carácter ", debe ponerse doble. Así, si se ejecutara

```
put_line("Hola␣""␣caracola");
```

el resultado debería ser

```
Hola " caracola
```

Cuando se necesiten caracteres especiales que requieran habitualmente estar "escapados", se deberán usar las funciones `int2char` y `char2int`. Así, si consideramos el siguiente código C++

```
char c;
c = '\n';
cout << "Hola\tCaracola\n";
```

podríamos escribir en ***alike*** uno que se comporta de la misma manera como sigue:

```
c: character;
c := int2char(10);
put_line("Hola",int2char(9),"Caracola",c)
```

19. El lenguaje dispone de funciones que devuelven datos escalares (`character`, `integer`, `boolean`). La devolución se lleva a cabo mediante la instrucción "return". La sintaxis es la que se muestra a continuación:

```
function valAbs(x:integer) return integer is
begin
    if (x < 0) then
        return -x;
    else
        return x;
    end if;
end;
```

Se considera un error que aparezca una instrucción "return" dentro de un procedimiento.

20. El lenguaje maneja también arrays, que se declaran con la siguiente sintaxis:

```
procedure datos(d: integer; w: array(1..100) of integer) is
    v: array(-3..3) of integer;
    w: array(0..3) of integer;
begin
    ...
    v(-1) := w(3);
    ...
end;
```

A diferencia de C o C++, en la declaración de un "array" no se indica su tamaño, sino el *rango de índices* de las componentes. Para que esté correctamente declarado, los índices tienen que ser constantes enteras, y el primer índice ha de ser menor o igual que el segundo. Así, si el rango de índices está declarado como `a..b`, cualquier valor entero i tal que $a \leq i \leq b$ será un índice correcto de la correspondiente componente del vector.

21. El lenguaje dispone del procedimiento sin parámetros `exit`. Su ejecución abandona la ejecución del programa.

Veamos un ejemplo de programa completo en ***alike*** (consúltese la batería de programas suministrada para tener una visión completa de todos los elementos del lenguaje):

```
procedure factorial is --el programa principal se llama "factorial"
    n: integer;
-----------------------------------------------------------------
function fact(m: integer) return integer is
begin
    if m = 1 then
        return 1;
    else
        return m * fact (m-1);
    end if;
end;
-----------------------------------------------------------------
begin
    put_line("Calcula␣k!␣para␣k=1..20.");
    put("Habrá␣MATH␣overflow␣en␣13!␣porque␣los␣enteros␣");
    put_line("en␣la␣máquina␣P␣son␣de␣4␣bytes.");

    n := 1;
    while n <= 20 loop
        put_line (n, "!␣=␣", fact (n));
        n := n + 1;
    end loop;
end;
```