El lenguaje *gcl*

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 ***gcl***, que manejaremos a lo largo de las sesiones de laboratorio del curso 24-25. ***gcl*** es un lenguaje basado en el *guarded command language*, propuesto por Edsger Dijkstra, con algunas extensiones y adaptaciones al español. Se trata de un lenguaje estructurado que contiene un conjunto de instrucciones y de estructuras básicas.

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: *caracter*[1], *booleano* y *entero*. La declaración de variables sigue una sintaxis análoga a la de C++:

   `<tipo_escalar><lista_de_identificadores>;`

   Por ejemplo:

   ```
   entero i, j, k;
   caracter c, d, e;
   booleano v;
   ```

   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.

[1] La palabra reservada se usará sin la tilde

4. El lenguaje es **case-insensitive**, tanto para las palabras reservadas 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 "entero", o "Mq", por ejemplo.

6. La asignación se realiza con el operador `:=`, y no se permite la asignación de un tipo de dato a otro distinto. Por otro lado, solo se permite la asignación de datos escalares, no de arrays.

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

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

8. El lenguaje maneja también arrays, que se declaran con la sintaxis

   `<tipo_escalar>[<num_componentes>] <lista_de_identificadores>;`

   Por ejemplo, la declaración de dos arrays de 10 enteros sería:

```
entero[10] w1, w2;
```

   Análogamente al caso de C++ o java, en la declaración de un "array" se indica su tamaño, siendo el rango de índices [0,tamaño-1]. Por lo tanto, es necesario que el tamaño sea un valor natural positivo.

9. Las condiciones de las instrucciones de selección (`Sel`) y de iteración (`Mq`), que se describen a continuación, sólo puede ser de tipo `booleano`.

10. La instrucción de selección se escribe como sigue, siendo la parte `dlc` (acrónimo de "*De Lo Contrario*") opcional:

```
Sel
    <expresión>:
        instrucciones --una o más instrucciones
    <expresión>:
        instrucciones --una o más instrucciones
    ...
    dlc:
        instrucciones --una o más instrucciones
FSel
```

    con las siguientes restricciones:

    - Debe haber al menos una estructura `<expresión>: instrucciones`
    - No puede haber una claúsula `dlc` si no hay al menos una estructura `<expresión>: instrucciones`

- La claúsula `dlc`, si la hay, debe ser la última de la instrucción.

Desde un punto de vista semántico, la selección funciona de la siguiente manera:

- Si ninguna guarda está abierta y no hay claúsula `dlc`, la instrucción no hace nada.
- Si ninguna guarda está abierta y hay claúsula `dlc`, se ejecutan las instrucciones de la claúsula `dlc` y la instrucción termina.
- Si hay varias guardas abiertas, se ejecutan las instrucciones de la primera guarda que se cumpla, y se acaba la selección.

11. La instrucción de iteración se escribe como:

```
Mq <expresión>
    instrucciones --una o más instrucciones
FMq
```

Su comportamiento es el habitual en un bucle `while`.

Como en el proceso de desarrollo a veces es interesante dejar un bloque vacío, que será completado en una fase posterior, ***gcl*** suministra la instrucción *nada*, que no tiene ningún efecto en la ejecución del programa.

```
Mq x>0
    nada; --a completar más tarde
FMq
```

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

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

14. 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 `Sel` o `Mq`.

15. Cuando un procedimiento o función no tiene parámetros, debe invocarse con paréntesis.

16. 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
inicializar(booleano[80] ref colonia1, colonia2; entero i1, i2)
    ...
Principio
    ...
Fin
```

La clase de parámetro (por valor o por referencia) se aplica a toda la lista de identificadores. Así, en el ejemplo anterior, tanto `colonia1` como `colonia2` son parámetros por referencia, mientras que `i1` e `i2` son parámetros por valor.

17. Las variables escalares, los parámetros simples, y las componentes de variables o de parámetros de tipo vector son asignables (pueden aparecer en la parte izquierda de una asignación o en un paso por referencia). Ni los procedimientos ni el programa principal son asignables.

18. Las funciones devuelven datos escalares (`caracter`, `entero`, `booleano`). No existe una instrucción `return`. La devolución del resultado por parte de una función se lleva a cabo mediante una instrucción de asignación especial, en que la parte izda. es el identificador de la propia función. Esta asignación no supone salir de la función (como sería el caso de una instrucción `return`), sino que el valor que finalmente devolverá la función será el último valor asignado en estas asignaciones especiales.

```
entero valAbs(entero x)
Principio
    Sel
        caso x < 0:
            valAbs := -x;
        dlc:
            valAbs := x;
    FSel
Fin
```

Un procedimiento se declara de manera análoga a una función, con la salvedad de que no hay que indicar el tipo del dato devuelto, ya que no se devuelve ningún dato.

19. Se permite la escritura de variables y expresiones simples (no de "arrays"), mediante las instrucciones `escribir` y `escribir_lin` (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 `verdadero` o `falso` en el caso de booleanos. También se permite la escritura de constantes de tipo cadena. La instrucción `escribir` requiere al menos un argumento; `escribir_lin` puede no tener argumentos.

20. La entrada de datos escalares no booleanos, se hace mediante la instrucción `leer`, y requiere al menos un parámetro en su invocación.

    La instrucción `leer_lin(...)`, además de leer los datos escalares indicados, salta caracteres de la entrada hasta que logra leer y saltar un *new line*. En este caso, es posible también usarla sin parámetros cuando solo se desea saltar hasta el principio de la siguiente línea.

    Ejemplos serían:

```
entero n;
caracter c,b;

leer_lin(n,c,b);
--se podría escribir también como
-- leer(n,c,b);
-- leer_lin();
```

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

```
escribir_lin("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

```
escribir_lin("Hola␣""␣caracola");
```

el resultado debería ser

```
Hola " caracola
```

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

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

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

```
caracter c;
c := entAcar(10);
escribir_lin("Hola",entAcar(9),"Caracola",c)
```

22. El lenguaje dispone de la instrucción `abandonar`. La instrucción `abandonar` termina la ejecución del programa.

23. Los operadores aritméticos (`+, -, *, /, mod`) sólo admiten operandos de tipo `entero`.

24. Los operadores booleanos (`&, |, !`) sólo admiten operandos de tipo `booleano`.

25. Los operadores relacionales (`>, >=, <, <=, =, <>`) sólo admiten operandos del mismo tipo.

Los elementos que falten de especificar se pueden deducir a partir de los ejemplos de la batería de tests suministrada.