Especificación del lenguaje *Space*

Alfredo Ibias Martínez

Daniel Loscos Barroso

Doble Grado Matemáticas - Ingeniería Informática

1. Presentación del lenguaje

Para guiarnos en la especificación del lenguaje y en la elección de los requisitos que queremos que cumpla vamos a emplear la siguiente motivación: Queremos crear un lenguaje pensado para su uso en misiones espaciales. Su utilidad principal será crear los sistemas de control y soporte de los satélites y naves lanzadas al espacio. Por este motivo hemos decidido llamarlo *Space*.

Deberá ser un lenguaje sencillo y ligero, que evite lo más posible la generación de errores, y que esté fuertemente orientado a la ejecución de órdenes de forma secuencial. Es especialmente importante la minimización de los errores pues estos, por pequeños que sean, pueden llegar a causar pérdidas millonarias.

Por lo tanto, este lenguaje será un lenguaje imperativo básico, con una sintaxis simple y pocas diferencias con *C*, para que se pueda empezar a usar rápidamente si ya se conoce este lenguaje de programación.

2. Requisitos del lenguaje

El lenguaje cumplirá los siguientes requisitos:

2.1. Identificadores y ámbitos de definición

El lenguaje permitirá la declaración de variables simples y de arrays de cualquier tipo, incluidos arrays de arrays. Permitirá también bloques anidados (con su tabla de símbolos correspondiente) y permitirá crear funciones y procedimientos para simplificar y mejorar la legibilidad de los programas.

No incluirá punteros ni registros para no permitir el acceso a la memoria dinámica ni las referencias a memoria. El objetivo es evitar errores que pongan en peligro la integridad de partes sensibles de la memoria. Tampoco permitirá clases, módulos ni cláusulas de importación para evitar errores derivados de la programación modular y para simplificar los programas. Si se quiere usar alguna función de biblioteca, esta deberá ser copiada y pegada directamente en el código del programa.

2.2. Tipos

Se tendrá declaración explícita del tipo de las variables y los tipos predefinidos: int, float, bool y char. Así como los tipos sin nombre formados por arrays pluridimensionales de estos tipos básicos.

El lenguaje contendrá operadores infijos, con distintas prioridades y asociatividades, para realizar operaciones con los distintos tipos básicos.

2.3. Instrucciones ejecutables

Space tendrá instrucciones de:

- Declaración de variable.
- Asignación, incluyendo arrays.
- Condicional con una y dos ramas.
- Bucle. Se tendrá el típico bucle while.
- Instrucción case con salto a cada rama en tiempo constante.
- Llamadas a procedimientos y funciones.

Lo que no se tendrá serán expresiones con punteros y nombres cualificados, ni instrucciones de reserva o liberación de memoria dinámica para evitar los problemas con punteros y memoria dinámica señalados anteriormente.

2.4. Errores

El compilador indicará el tipo de error y acto seguido parará la compilación.

3. Sintaxis

La sintaxis del lenguaje se define a continuación:

3.1. Unidades Léxicas

Estas son las unidades léxicas que hemos contemplado:

$simbolo = (+|-|*|/|=|>|<|>=|<=|==|!=|\wedge|\%|\&\&|"||"|$
$|(|)|\{\backslash n|\{|\}\backslash n|\};\backslash n|;|,\backslash n|,|[|]|:\backslash n|:)$
$letraMin = [a-z]$
$letraMay = [A-Z]$
$letra = (\{letraMin\}|\{letraMay\})$
$digitoPositivo = [1-9]$
$digito = (\{digitoPositivo\}|0)$
$parteEnt = \{digitoPositivo\}\{digito\}*$
$parteDec = \{digito\} * \{digitoPositivo\}$
$int = (\{parteEnt\}|0)$
$real = (\{int\}|\{int\}.\{parteDec\})$
$char =' (\{letra\}|\{digito\}|\{simbolo\}|\{separador\}|\backslash n)'$
$bool = (true|false)$
$palabrasReservadas = (DECLARE\backslash n|proc|bool|char|int|float$
$|DEFINE\backslash n|return|START\backslash n|END|if|else|while|switch|case|default)$
$separador = [\ \backslash t\backslash r]$
$saltodelinea = [\backslash n]$
$comentario = //(\{letra\}|\{digito\}|\{simbolo\}|\{separador\}|.) * \{saltodelinea\}$
$idVar = \{letraMin\}(\{digito\}|\{letra\}|_)*$
$idFun = \{letraMay\}(\{digito\}|\{letra\}|_)*$

3.2. Generalidades

Todo programa empieza en START y ejecuta las instrucciones que encuentra hasta llegar a END. Sin embargo el código no empieza ahí, tendremos tres secciones de código:

- DECLARE: Es el primer bloque de código, en él se declaran las funciones y los procedimientos.

- DEFINE: Es la sección del código donde se definen las funciones y procedimientos que se usarán en MAIN.
- MAIN: Empieza por START y acaba en END, es el bloque que contiene las instrucciones que van a ejecutarse, incluyendo la declaración de variables. La palabra reservada END cierra el código.

De este modo, un archivo, punto de entrada a la gramática, tendría esta forma:

$$ARCHIVO \rightarrow DECLARE \quad DEFINE \quad MAIN$$

3.3. Cierre de expresiones y bloques

La mayoría de las instrucciones acaban con ";", salvo las de control.
Tendremos distinto tipos de bloques:

- Los bloques de instrucciones están delimitados por llaves (elementos "{" y "}") y se permite que estén anidados de forma que las variables declaradas en un bloque no afectan al bloque que las contiene, pero si se modifica una variable que ya existía, la modificación afecta al bloque padre. Las instrucciones que generan nuevos bloques de instucciones son las de control.

  $BLOQUEI \rightarrow \{ \quad INSTRUCCIONES \quad \}$
  $INSTRUCCIONES \rightarrow INSTRUCCION\backslash n \quad INSTRUCCIONES|\varepsilon$
  $INSTRUCCION \rightarrow DECVAR|ASIG|IF|IFELSE|WHILE|SWITCH|PROC$

- El bloque de declaración de funciones y procedimientos empezará por DECLARE y contendrá únicamente las cabeceras. Los identificadores de funciones y procedimientos ($IDFP$) empezarán por mayúscula y contendrán solo letras, dígitos o el carácter _.

  $DECLARE \rightarrow \text{DECLARE}\backslash n \quad CABECERAS$
  $CABECERAS \rightarrow CABECERA\backslash n \quad CABECERAS|\backslash n$
  $CABECERA \rightarrow CABECERAF|CABECERAP$
  $CABECERAF \rightarrow TIPO \quad IDFP \quad (TARGS);$
  $CABECERAP \rightarrow \text{proc} \quad IDFP \quad (TARGS);$
  $TIPO \rightarrow \text{bool|char|int|float}$
  $TARGS \rightarrow TIPO|TIPO, \quad TARGS$

- El bloque de definición de funciones y procedimientos empezará por DEFINE y contendrá los bloques de implementación de funciones y de procedimientos:

  $DEFINE \rightarrow \text{DEFINE}\backslash n \quad IMPLEMENTACIONES$
  $IMPLEMENTACIONES \rightarrow IMPLEMENTA\backslash n \quad IMPLEMENTACIONES|\backslash n$
  $IMPLEMENTA \rightarrow IMPLEMENTAF|IMPLEMENTAP$

- Los implementación de funciones:

  $IMPLEMENTAF \rightarrow IDFP \quad (ARGS)\{INSTRUCCIONES \quad RETURN\}$
  $ARGS \rightarrow ID|ID, \quad ARGS$
  $RETURN \rightarrow$ return $\quad EXPRESION;$

- Los implementación de procedimientos:

  $$IMPLEMENTAP \rightarrow IDFP \quad (ARGS)BLOQUEI$$

- Finalmente, el bloque principal de ejecución:

  $$MAIN \rightarrow \text{START}\backslash\text{n} \quad INSTRUCCIONES \quad \text{END}$$

3.4. Expresiones

Las expresiones podrán contener:

- Identificadores de variable: *ID*
- Llamadas a función: *IDFP (ARGS)* nombreFunción(var1, var2, ..., varN)
- Numeros, booleanos o caracteres: *ELEM*
- Operadores:
  - Suma: + (infijo, entre dos ints o floats, 1).
  - Resta: - (infijo, entre dos ints o floats, 1).
  - Menos: - (prefijo a un int o float, 4).
  - Multiplicación: * (infijo, entre dos ints o floats, 2).
  - División: / (infijo, entre dos ints o floats, 2).
  - Módulo: % (infijo, entre dos ints, 2).
  - Potencia: $\wedge$ (infijo, entre dos ints o floats, 3).
- Comparadores:
  - Comparación de igualdad: == (infijo, entre dos tipos iguales, 0).
  - Comparación de desigualdad: != (infijo, entre dos tipos iguales, 0).
  - Comparación de menor o igual: <= (infijo, entre dos ints, floats o chars, 0).
  - Comparación de mayor o igual: >= (infijo, entre dos ints, floats o chars, 0).
  - Comparación de menor: < (infijo, entre dos ints o floats, 0).

- Comparación de mayor: $>$ (infijo, entre dos ints o floats, 0).
- AND lógico: && (infijo, entre dos tipos booleanos, 2).
- OR lógico: || (infijo, entre dos tipos booleanos, 1).
- NOT lógico: $\sim$ (prefijo a un tipo booleano, 4).

Las expresiones de módulo y potencia son nativas a nuestro lenguaje. Modificaremos la máquina-P para que las soporte.

$$
\begin{aligned}
&EXP \rightarrow E1 \quad OP0 \quad EXP | E1 \\
&E1 \rightarrow E2 \quad OP1 \quad E1 | E2 \\
&E2 \rightarrow E3 \quad OP2 \quad E2 | E3 \\
&E3 \rightarrow E4 \quad OP3 \quad E3 | E4 \\
&E4 \rightarrow OP4 \quad E4 | E5 \\
&E5 \rightarrow ID | IDFP(ARGS) | ELEM | (EXP) \\
&OP0 \rightarrow == | != | <= | >= | < | > \\
&OP1 \rightarrow + | - | || \\
&OP2 \rightarrow * | / | \% | \&\& \\
&OP3 \rightarrow \wedge \\
&OP4 \rightarrow - | \sim
\end{aligned}
$$

3.5. Declaración de variables

Las variables de tipos básicos se declaran poniendo el tipo de la variable seguido del identificador de la misma.

Los identificadores empiezan siempre con minúscula. Los arrays se indican en la declaración de la variable añadiéndoles, después del tipo, un subíndice que indica su longitud (número entero entre corchetes "[" y "]"). Al usarlos en expresiones, se debe incluir después del identificador de la variable, el subíndice concreto que se está usando del array. Para referirse al array entero (por ejemplo en un paso de parametros a una función) se usa el identificador de la variable sin los corchetes. En uso todos los arrays tiene como indice mínimo 0 y como máximo su dimensión menos uno.

$$
\begin{aligned}
&DECVAR \rightarrow TVAR | SVAR \\
&TVAR \rightarrow TIPO \quad DIMS \quad ID; \\
&DIMS \rightarrow DIM | DIM \quad DIMS \\
&DIM \rightarrow [EXP] | \varepsilon
\end{aligned}
$$

3.6. Asignación

La asignación de variables se hace con su identificador, el símbolo = y la expresión que queremos asignarle.

$$ASIG \rightarrow ID \quad = \quad EXP;$$

En ningún caso se admitirá la asignación en el mismo momento de la declaración de una variable.

3.7. Instrucciones de control

Las instrucciones de control habilitadas son:

- Condicional con una rama. Su estructura es:

  if (condición) {
  // Rama 1.
  }

  $$IF \rightarrow \text{if}(EXP) \quad BLOQUEI$$

- Condicional con dos ramas. Su estructura es:

  if (condición) {
  // Rama 1.
  } else {
  // Rama 2.
  }

  $$IFELSE \rightarrow IF \quad \text{else} \quad BLOQUEI$$

- Bucle While. Su estructura es:

  while (condición) {
  // Contenido del bucle.
  }

  $$WHILE \rightarrow \text{while}(EXP) \quad BLOQUEI$$

- Condicional Case. Su estructura es:

```
switch EXP
case INDICE: {
    // Rama1.
}
    ⋮
default: {
    // Rama por defecto (puede estar vacía).
}
```

Los indices de los cases empezarán siempre en 1 y serán enteros consecutivos.

$$SWITCH \rightarrow \text{switch}(EXP) \quad CASES \quad \text{default:} \quad BLOQUEI$$
$$CASES \rightarrow \text{case } EXP\text{:} \quad BLOQUEI \quad CASES|\varepsilon$$

3.8. Llamadas a procedimientos

Los procedimientos se llaman directamente con su identificador y los parámetros: nombreProc(var1, var2, ..., varN);

$$PROC \rightarrow IDFP(ARGS);$$

Tanto para procedimientos como para funciones, todos los argumentos se pasan por valor.

4. La máquina-PSpace

El lenguaje *Space* esta pensado para compilarse y generar código-PSpace. Es muy parecido al código-P de la máquina-P pero con una ligera modificación:

La máquina-PSpace admite como nativas las instrucciones de potencia y módulo. Construimos la máquina-PSpace modificando el código de la máquina-P aportada por el profesor para añadirle estas operaciones.

La máquina-PSpaces solo trabaja con tipos de datos enteros y booleanos. Los tipos float y char se convierten en enteros por truncamiento o indice ascii para poder procesarse.