Especificación del lenguaje *Space*

Alfredo Ibias Martínez

Daniel Loscos Barroso

Doble Grado Matemáticas - Ingeniería Informática

Trabajo realizado con LaTeX

Índice

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 (integer, real, boolean, character, Numeric y Type).

El lenguaje contendrá operadores infijos, con distintas prioridades y asociatividades, para realizar operaciones con los distintos tipos. Se podrá tener también tipos sin nombre y se podrán definir tipos de usuario de forma limitada (en concreto, sólo enumerados).

La equivalencia de tipos será por nombres, para evitar problemas.

2.3. Instrucciones ejecutables

Space Tendrá instrucciones de:

- 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.
- Expresiones formadas por constantes, identificadores (con y sin subíndice) y operadores infijos.
- 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 las fila y columna donde se ha producido. 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:

$letraMin = \text{a}|...|\text{z}$

$letraMay = \text{A}|...|\text{Z}$

$letra = letraMay|letraMin$

$digitoPos = 1|...|9$

$digito = digitoPos|0$

$parteEnt = digitoPos(digito)^*$

$parteDec = (digito)^*digitoPos$

$int = parteEnt|0$

$real = int|(int).parteDec$

$separador = \backslash\text{t}|\backslash\text{n}$

$simbolo = +|-|=|>|\,\%|...|\backslash|.|,|;$

$comentario = \backslash\backslash(letra|digito|\backslash\text{t}|simbolo)^*\backslash\text{n}$

$identificadorVar = letraMin(digito|letra|\_)^*([parteEnt|0])^*$

$char = \text{'}(letra|digito|simbolo|separador)\text{'}$

$num = int|real$

$bool = true|false$

$type = num|char|bool$

$argumento = type|identificadorVar$

$identificadorFun = letraMay(digito|letra|\_)^*((\varepsilon|(argumento(,argumento)^*))$

$identificador = identificadorVar|identificadorFun$

$palabrasReservadas = \text{while|if|else|...|float|bool|int|...|return|DEFINE|DECLARE|START|END}$

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 variables globales, 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 y 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

Las expresiones se acaban con ";" para determinar donde acaba la expresión. Alternativamente, el final de un bloque lleva implícito el ";".

Los bloques están delimitados por llaves (elementos "{" y "}") y se permite que estén anidados.

Para evitar errores, este lenguaje no permitirá que los ";" y las llaves "{" y "}" estén en una linea distinta a la que deberían estar. Luego, si se detecta un salto de linea cuando debería haber uno de estos símbolos, saltará un error y no compilará, aunque éste esté en la linea siguiente. Esto conlleva que no se permiten las instrucciones multilineas para mejorar la claridad del código, sino que se deberá poner todo en la misma línea.

3.4. Comentarios

Los comentarios sólo pueden ser de una linea: empiezan con "//" y llegan hasta el final de línea.

3.5. Declaración de variables y constantes

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

Los identificadores de variables y funciones empiezan siempre por letra, ya sea mayúscula o minúscula.

Los arrays se indican en la declaración de la variable poniendo después del tipo "[longitud]", donde "longitud" debe de ser un número. Al usarlos en expresiones, se debe incluir después del identificador de la variable, entre corchetes, el índice 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.

Un array de varias dimensiones lo que tiene son varios corchetes, uno por dimensión, después del identificador de variable (o del tipo cuando se declara la variable).
Finalmente, se pueden declarar constantes añadiendo a la declaración de tipos la palabra "const" justo antes del tipo.

3.6. Expresiones

Las expresiones podrán contener:

- Identificadores: en su primera aparición se les debe asignar un tipo poniendo el nombre del mismo antes del identificador de la variable.
- Operadores:
  - Suma: + (infijo, entre tipos numerales).
  - Resta: - (infijo, entre tipos numerales).
  - Multiplicación: * (infijo, entre tipos numerales).
  - División: / (infijo, entre tipos numerales).
  - Módulo: % (infijo, entre tipos numerales).
  - Potencia: ^ (infijo, entre tipos numerales).
  - Concatenación: + (infijo, entre cadenas de carácteres).
- Comparadores:
  - Comparación de igualdad: == (infijo).
  - Comparación de desigualdad: != (infijo).
  - Comparación de menor o igual: <= (infijo, entre tipos numerales).
  - Comparación de mayor o igual: >= (infijo, entre tipos numerales).
  - Comparación de menor: < (infijo, entre tipos numerales).
  - Comparación de mayor: > (infijo, entre tipos numerales).
  - AND lógico: && (infijo, entre tipos booleanos).
  - OR lógico: || (infijo, entre tipos booleanos).
- Asignaciones: usando = (infijo).
- Asignaciones compuestas: usando op= (infijo) donde op es una operación aplicable al tipo de la asignación.

Todas estas expresiones funcionan en resumen como lo hacen en cualquier otro lenguaje como C o Java.

3.7. Instrucciones de control

Las instrucciones de control habilitadas son:

- Condicional con una rama. Su estructura es:
  if (condición) {
  // Rama 1.
  }

- Condicional con dos ramas. Su estructura es:
  if (condición) {
  // Rama 1.
  } else {
  // Rama 2.
  }

- Bucle While. Su estructura es:
  while (condición) {
  // Contenido del bucle.
  }

- Bucle Do-While. Su estructura es:
  do {
  // Contenido del bucle.
  } while (condición);

- Bucle For. Su estructura es:
  for (integer i = 0; condición(i); i = i + 1) {
  // Contenido del bucle.
  }

- De salto. Su estructura es: break;
  Simplemente salta fuera del bloque de control en el que se encuentra.

- Condicional Case. Su estructura es:
  switch elemento {
  case comparación1: // Rama 1.
  break;
  ⋮
  default: // Rama por defecto.
  break;
  }

- De retorno: su estructura es: return;
  Acaba una función y señala el valor que esta devuelve.

3.8. Funciones y procedimientos

Las funciones y procedimientos se tratan igual, la diferencia es que un procedimiento es una función que devuelve un valor de tipo void. La sintaxis es:

tipoDeRetorno nombreFunción(tipoVar1 var1, tipoVar2 var2, ... tipoVarN varN) {
// Cuerpo de la función.
}

Para volver de una función o procedimiento se usa la instrucción return. Las que devuelven algún valor necesitan que en return se especifique la variable que contiene el valor (o el valor directamente) que se devuelve.

3.9. Tipos

Los tipos predefinidos son:

- Tipo número entero: int.
- Tipo número en coma flotante: float.
- Tipo expresión lógica: bool.
- Tipo carácter: char.
- Tipo cadena de carácteres: string.
- Tipo vacío: void.

Para que un usuario defina sus propios tipos sólo se permite la creación de enumerados, con la siguiente estructura:

enum nombreDelEnumerado {
elemento1,
elemento2,
⋮
elementoN
}

Estos enumerados sirven principalmente para identificar una serie de valores numéricos con unos identificadores más explícitos y entendibles (como todos los enumerados).

4. Ejemplos de programas

A continuación se muestran una serie de ejemplos de programas típicos realizados con este lenguaje.[1]

4.1. Hola Mundo

Al no tener un sistema de entrada/salida, el hola mundo consistirá en escribir dicha frase en una cadena.

```
    int main () {
string cadena;
cadena = "Hola mundo";
return 0;
}
```

4.2. Suma

```
    int main () {
int a = 10;
int b = 5;
int suma = a + b;
return 0;
}
```

4.3. Potencia (con función)

```
    int main () {
int a = 10;
int b = 2;
int pow = pow(a, b);
return 0;
}

    int pow(int a, int b) {
return a ^ b;
}
```

[1] A todos los programas les falta la identación porque no he sido capaz de que me funcione en LaTeX

4.4. Factorial

```
    int main () {
int a = 0;
int f = 10;
a = f - 1;
while (a >0) {
f = factorial(a, f);
a = a - 1;
}
return 0;
}

    int factorial(int a, int f) {
return f * a;
}
```

4.5. Programa estandar

```
    int main () {
// Declaración de variables
while (true) {
if (func1) {
funcRespuesta1;
}
if (func2) {
funcRespuesta2;
}
⋮
if (funcN) {
funcRespuestaN;
}
}
return 0;
}
```