Especificación del lenguaje Space

Alfredo Ibias Martínez

Doble Grado Matemáticas - Ingeniería Informática

Trabajo realizado con LaTeX

Índice

1. Motivación

Para la especificación del lenguaje, y en especial a la hora de escoger los requisitos que va a cumplir, voy a buscar una motivación para crear dicho lenguaje, con el objetivo de justificar las elecciones tomadas a la hora de especificarlo.

La motivación que voy a seguir es la de suponer que el lenguaje que voy a crear es un lenguaje pensado para el uso en misiones espaciales. Es decir, cuyo uso principal será crear los sistemas de control y ejecución de los satélites y naves lanzadas al espacio. Por este motivo, he dedicido llamarlo Space.

Esto supone que este lenguaje debe 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, reduciendo la paralelización al mínimo necesario. Es especialmente importante la minimización de los errores pues estos, por pequeños que sean, pueden llegar a causar pérdidas millonarias, ya que todo el proceso involucrado en la investigación espacial, y en especial en el lanzamiento de aparatos al espacio, es bastante costoso.

Por tanto, este lenguaje será un lenguaje imperativo básico. Aparte, será necesario que tenga una sintaxis relativamente conocida, pues la idea es que sea ampliamente aceptado y que su curva de aprendizaje sea la menor posible, para que se pueda empezar a usar rápidamente si ya conoces algún otro lenguaje de programación y que su adopción no suponga un coste extra.
Por lo tanto no va a ser un lenguaje muy original y se podría entender como un lenguaje C restringido/simplificado, cuyo objetivo es evitar la mayor cantidad posible de errores.

Es cierto que esto supone que el lenguaje tendrá que funcionar por encuesta más que por callbacks (como se verá más adelante), aumentando su tiempo de respuesta. Pero teniendo en cuenta que el tiempo no suele ser una cuestión crucial en las misiones espaciales (al menos no al límite de las milésimas de segundo), el hecho de que la mayoría de programas en su función main vayan a tener un bucle no debería ser problema (de hecho está pensado para eso).

Finalmente, una parte importante de este lenguaje sería un sistema de entrada/salida para poder comunicarse con el resto de sistemas, con ficheros y con los usuarios. Sin embargo, dadas las limitaciones de la práctica y la complejidad que esto añade al lenguaje, en un principio esto no se tendrá en cuenta y se valorará la posibilidad de añadir al final, si hay tiempo, un sistema básico de comunicación mediante consola.

2. Requisitos del lenguaje

El lenguaje cumplirá los siguientes requisitos:

2.1. Identificadores y ámbitos de definición

A este respecto, el lenguaje permitirá la declaración de variables simples y de arrays de cualquier tipo, incluidos arrays de varias dimensiones (que son equivalentes a los arrays de arrays).

Permitirá también bloques anidados (con su tabla de símbolos correspondiente), y permitirá funciones y procedimientos para simplificar y mejorar la legibilidad de los programas.

No permitirá punteros ni registros, para no permitir el acceso a la memoria dinámica ni las referencias a memoria, y por tanto el posible acceso por algún error a otras partes de la memoria que sean fundamentales. Esto es especialmente importante cuando se sabe que los registros o la memoria pueden ser modificados por la interacción con cualquier elemento externo que los dañe o modifique.

Tampoco permitirá clases, módulos ni cláusulas de importación para evitar errores derivados de la programación modular, y para simplificar los programas al evitar el uso de bibliotecas generales que contienen más código que el que se va a usar realmente. Si se quiere usar alguna función de biblioteca, esta deberá ser copiada y pegada directamente en el código del programa.

2.2. Tipos

A este respecto se tendrá declaración explícita del tipo de las variables, y se tendrán tipos predefinidos (integer, float, boolean, character, string y void).

Se tendrán también operadores infijos, con distintas prioridades y asociatividades, para realizar operaciones con los distintos tipos.

Se podrán tener también tipos sin nombre y con nombre, y se podrán definir tipos de usuario de forma limitada (en concreto, sólo enumerados, es decir, realmente no se podrán definir nuevos tipos).

Finalmente, la equivalencia de tipos será por nombres, para evitar problemas derivados de tener dos tipos con la misma definición pero distinto nombre.

2.3. Instrucciones ejecutables

A este respecto es tendrán instrucciones de:

- Asignación, incluyendo arrays. Se usará para ello el símbolo = infijo.
- Condicional con una y dos ramas. Se usará para ello la estructura if/else.
- Bucle. Se tendrá tanto el típico bucle while como los bucles for y do/while.
- Instrucción case con salto a cada rama en tiempo constante.
- Expresiones formadas por constantes, identificadores con y sin subíndice, y por 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

A este respecto, el compilador indicará el tipo de error, y las fila y columna donde se ha producido, e intentará proseguir con la compilación tras detectar un error para detectar más errores.

3. Sintaxis

La sintaxis del lenguaje se define a continuación:

3.1. Generalidades

Todo programa empieza en la función main, que será una función que devuelve un entero, a imagen de las funciones main de la mayoría de lenguajes conocidos.
Los paréntesis (elementos "(" y ")") modifican la asociación de los operandos.
Los números se escriben tal cual en el código, pero los carácteres deben ir entre comillas simples y las cadenas de carácteres entre comillas dobles.

3.2. 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.

3.3. Comentarios

Los comentarios pueden ser de una linea (empiezan con "//" y llegan hasta el final de línea) o de varias líneas (empiezan con "/*" y acaban con "*/") y se permite anidación de comentarios.

3.4. 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.

3.5. 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.

3.6. 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.

3.7. 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.

3.8. 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;
}
```