Memoria del lenguaje *Space*

Alfredo Ibias Martínez

Daniel Loscos Barroso

Doble Grado Matemáticas - Ingeniería Informática

Es 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.

1. Características del lenguaje

El lenguaje cumplirá los siguientes requisitos:

1.1. 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.

La memoria está optimizada de modo que siempre se reservan exactamente el máximo número de direcciones que necesita el marco simultáneamente.

1.2. 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 tipos básicos.

1.3. Instrucciones ejecutables

Space tendrá instrucciones de: Declaración de variable (tipo id;), Asignación (id = exp;), Condicional con una y dos ramas (if (exp) else), Bucle (while(exp)), Case con salto a cada

rama en tiempo constante (switch(exp) case1: .... default) donde la existencia de un default es necesaria y Llamadas a procedimientos (procname(exp,exp,exp,exp)).

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

1.4. Errores

El compilador indicará el tipo de error, la linea y la columnda de mismo. Acto seguido parará la compilación.

2. Sintaxis

Todo programa empieza en START y ejecuta las instrucciones que encuentra hasta llegar a END. Tendremos tres secciones de código: DECLARE: donde se declaran las funciones y los procedimientos. DEFINE: donde se definen las funciones y procedimientos que se usarán en MAIN. MAIN: Empieza por START y acaba en END, con todas las instrucciones que se van a ejecutar. La palabra reservada END cierra el código y el archivo.

2.1. Cierre de expresiones y bloques

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

- El bloque principal delimitado por BEGIN y END.
- 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 instrucciones son las de control.
- 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 _.
- El bloque de definición de funciones y procedimientos empezará por DEFINE y contendrá los bloques de implementación de funciones y de procedimientos. Los dos son secuencias de instrucciones; terminadas en un return obligatorio para las funciones.

2.2. Expresiones

Las expresiones básicas serán: Identificadores de variable, llamadas a función, números, booleanos y caracteres: Los operadores con sus prioridades serán:

- 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).
- 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. Modificamos la máquina-P para que las soportase.

2.3. 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.

3. 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.

4. Estructura del Compilador

Se le llama por consola con dos parámetros: el fichero de input y el de output.

Para compilar el input lo primero que hacemos es utilizar Jlex y CUP para construir el árbol de sintaxis abstracta.

El resto del proceso son sucesivos recorridos sobre el árbol que lo van decorando, comprobando errores y preparando para la generación de código: identificación de identificadores, comprobación de tipos, reserva y asignación de direcciones de memoria para las variables y cálculo de los saltos de instrucciones para la generación de código.

Finalmente realizamos un último recorrido que genera el código máquina para la máquina-PSpace.