Diseño de Lenguajes de Programación
3º Curso de Ingeniería Informática del Software
Escuela de Ingeniería Informática (Oviedo)
Universidad de Oviedo

Introducción

Objetivo

La práctica obligatoria de la asignatura consiste en el diseño, implementación y documentación de un lenguaje de programación aplicando las técnicas de construcción de procesadores de lenguajes vistas a lo largo de las clases de teoría.

Versiones de la Práctica

Existen dos versiones de la práctica. Hay una versión reducida que será la implementada por los alumnos que opten por la evaluación continua y una versión normal para aquellos que opten por alguna de las otras convocatorias.

El sistema de evaluación de ambas versiones de la práctica será el mismo. Se presentará a un examen práctico donde el alumno deberá realizar modificaciones a su lenguaje y posteriormente probar su correcto funcionamiento implementando un programa con el mismo.

En este documento se describe *únicamente la versión reducida* de la práctica. La versión completa será descrita en otro documento posterior.

Ámbito de este enunciado

Cada grupo de prácticas tiene variaciones en el lenguaje que debe entregar. Si el alumno no ha adquirido este documento en su grupo de prácticas no debe usar este documento como especificación de la misma. El alumno es responsable de realizar la práctica del grupo al que pertenece asegurándose de tener la especificación y el programa de ejemplo que le corresponde. No se aceptarán prácticas de un grupo que no le corresponde al alumno.

Cuestiones de implementación

La práctica deberá realizarse de manera individual no permitiéndose prácticas realizadas en grupo.

La práctica se realizará en el lenguaje Java. Queda a decisión del alumno la elección de la plataforma (Windows, Unix, Mac, ...) y el entorno de desarrollo para la construcción de la práctica.

La práctica deberá implementarse utilizando las herramientas de construcción de compiladores vistas en clase (jflex y byaccj). Al utilizar el generador de analizadores sintácticos “byaccj” la gramática no podrá tener ningún conflicto “shift/reduce” ni “reduce/reduce”.

Descripción del Lenguaje de entrada al Compilador

Programa de Ejemplo

En este apartado se describirán las características del lenguaje de entrada al compilador. Este documento amplía el detalle sobre las características del lenguaje que se mostraron mediante un programa de ejemplo (*ejemplo.txt*) durante la clase práctica correspondiente. Por tanto

deben tomarse como requisitos de la práctica, además de los incluidos en este documento, ***aquellos que mostrara dicho ejemplo***.

Requisitos

Aspectos léxicos

- El lenguaje debe tener los comentarios de C/C++: /**/ y //.
- Deberá tener constantes de tipo carácter (entre comillas simples), entero y real.
- Los identificadores deberán admitir la letra ñ (mayúsculas y minúsculas).

Tipos simples o primitivos

- Entero, float y char/byte (estos dos últimos son sinónimos; se debe elegir uno de los dos nombres)

Definición de variables

- No habrá definición múltiple de variables (separadas por comas).
- Se permitirá la definición de variables globales y locales tanto simples como compuestas (arrays y estructuras).
- Las variables globales podrán estar definidas en cualquier parte del fichero (antes, después o entre las funciones) y serán visibles únicamente en las funciones definidas posteriormente.
- No es obligatorio que haya variables globales definidas en el fuente

Asignación

- No hay conversión implícita al tipo de la expresión de la izquierda.
- No habrá asignación múltiple (por ejemplo 'a = b = 0;')

Conversiones de Expresiones

- No habrá conversiones implícitas.
- Se podrán realizar conversiones explícitas entre los tipos primitivos (char, int y real) mediante un operador de conversión (también llamado *cast*). En este caso el tipo a convertir deberá ser distinto al tipo de la expresión.

Los operadores que deberá incluir el lenguaje son:

- Aritméticos: +, -, * y /. Aplicables a enteros y float.
- Comparación: mayor, mayor o igual, menor, menor o igual, igual y distinto. Aplicables a enteros y reales.
- Lógicos: && (and), || (or) y ! (not). Aplicables a enteros.

Sentencias de control de flujo

- Sentencia condicional IF con ELSE opcional
- Sentencia WHILE
- En ninguna de ellas es obligatorio que haya sentencias dentro de las llaves.
- Los paréntesis alrededor de la condición y las llaves alrededor de las sentencias son obligatorios (aunque solo haya una sentencia).

Entrada y Salida

- Sentencias de lectura y escritura por la E/S estándar para todos los tipos primitivos con las instrucciones *read* y *print*.

Arrays

- Se permitirá declarar arrays multidimensionales (tanto de tipos simples como de tipos compuestos).
- En la definición de un array el tamaño es obligatorio y deberá ser una constante entera.
- Se permitirá acceder a los elementos de un array mediante una sintaxis de indexación. Para acceder al elemento de un array se usarán expresiones enteras (es decir, no tienen por qué ser solamente constantes sino cualquier expresión de dicho tipo). Ejemplo:

```
v[i+5] = v[func(x) + persona.edad];
```

- No hay que generar código que compruebe el acceso fuera de rango al array.

Estructuras/Registros

- La sintaxis de éstos es la definida en las clases prácticas.
- En las estructuras se podrán definir campos tanto de tipo simple como de tipos compuestos.
- Se permitirá acceder a los elementos de una estructura mediante el operador punto.
- La declaración de un nuevo tipo estructura solo podrá aparecer a nivel global (fuera de funciones y otras estructuras) y podrá aparecer antes, después o entre las funciones.

El lenguaje permitirá la definición e invocación de funciones.

- Los parámetros y el valor de retorno solo podrán ser de tipos simples.
- Todos los parámetros se pasan por valor.
- No habrá conversiones implícitas de los tipos de los argumentos a los tipos de los parámetros.
- Tampoco habrá conversión implícita del tipo del valor de retorno al tipo de retorno de la función.

- Si una función tiene tipo de retorno se deberá comprobar sus return tengan una expresión del mismo tipo. Si no lo tiene, sus return no deben tener valor de retorno.
- Una función podrá declarar variables locales solo al inicio de su cuerpo. Por tanto no podrá haber declaración de variables locales en cualquier lugar de la función ni dentro de bloques anidados (por ejemplo el creado por un while).
- La ejecución de un programa supone la invocación a su función "main" independientemente de dónde se encuentre.
- Una función podrá invocarse a sí misma mediante recursividad directa.
- No es obligatorio que haya funciones en un fichero fuente.
- No habrá sobrecarga de funciones.
- Una función solo podrá invocarse después de haber sido definida. Por tanto no podrá llamarse a una función definida más abajo en el fichero fuente.