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 e implementación de un lenguaje de programación aplicando las técnicas de construcción de Procesadores de Lenguajes vistas a lo largo de las clases de la asignatura.

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.

Requisitos 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 (Eclipse, Netbeans, Visual Studio Code, etc.).

La práctica deberá implementarse utilizando las técnicas, metalenguajes y herramientas de construcción de compiladores vistas en clase (*ANTLR4* y *MAPL*).

El alumno deberá añadir *únicamente* las características del lenguaje solicitadas. Cualquier añadido al lenguaje no solicitado (sentencias, operadores, etc.) puntuará negativamente o incluso podría invalidar la práctica.

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 muestra dicho ejemplo***.

Requisitos

Aspectos léxicos

- Deberá permitir constantes literales de tipo entero y real. Los literales de tipo real, además de una parte entera, deben tener obligatoriamente algún digito decimal.
- Deberá permitir constantes de tipo carácter (se usarán comillas simples). Se debe admitir la secuencia de escape *'\n'* para representar al carácter de salto de línea.
- El lenguaje debe tener los comentarios de C/C++: /**/ y //.
- El fichero de entrada se asume en UTF-8.

Tipos simples o primitivos

- Habrá tres tipos primitivos: entero (int), real (float) y carácter (char).

Definición de variables

- En la misma definición de la variable no se la podrá asignar un valor.
- 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.
- No habrá definición múltiple de variables (separadas por comas).

Asignación

- Sólo se pueden asignar valores de tipos simples.
- 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 (entre '<' y '>') deberá ser *distinto* al tipo de la expresión. Los paréntesis alrededor de la expresión son obligatorios. Ejemplo:

```
<int>(5.6)          // paréntesis obligatorios
```

Los operadores que deberá incluir el lenguaje son:

- Aritméticos: +, -, * y /. Aplicables a enteros y float. Operador módulo % para enteros.
- Comparación: mayor, mayor o igual, menor, menor o igual, igual y distinto. Aplicables a enteros y reales.
- Lógicos: && (and), || (or) y ! (not). Aplicables sólo a enteros y con evaluación sin corte.

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 no haya sentencias).

Entrada y Salida

- Sentencias de lectura y escritura por la E/S estándar para todos los tipos primitivos con las instrucciones *read* y *print* (tipo *char* incluído).
- La sentencia de escritura tendrá tres variantes:
    - Si se usa *print*, la siguiente escritura se realizará justo donde acabó la anterior.
    - Si se usa *printsp*, después del valor de la expresión, se imprimirá automáticamente un espacio en blanco.
    - Si se usa *println*, después del valor de la expresión, se imprimirá automáticamente un salto de línea.

Ejemplo	print 1; print 2;	printsp 1; print 2;	println 1; print 2;
Salida	12	1 2	1 2

    - En estas tres variantes, ***no es obligatorio que haya una expresión***. En ese caso, *printsp* y *println* imprimirán sólo el espacio o el salto de línea respectivamente y *print* no hará nada.

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).

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

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

Estructuras/Registros

- Un tipo estructura solo podrá ser usado debajo de su declaración.
- La sintaxis de éstos es la definida en las clases prácticas.
- Se permitirá acceder a los elementos de una estructura mediante el operador punto.
- En una estructura se podrán definir campos tanto de tipo simple como de tipos compuestos. No es obligatorio que haya algún campo.
- 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.

- Sólo dentro de las funciones puede haber sentencias (no puede haber sentencias a nivel global).
- Todos los parámetros se pasan por valor.
- Los parámetros y el valor de retorno solo podrán ser de tipos simples.
- 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 que sus *return* tengan una expresión del mismo tipo. Si no tiene tipo de retorno, 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, no podrá haber dentro de un *while*).
- Una función podrá invocarse a sí misma mediante recursividad directa.
- No es obligatorio que haya funciones en un fichero fuente. Por tanto, un programa vacío sería un programa válido.
- 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.
- No es obligatorio que una función tenga sentencias.
- Se puede llamar, como una sentencia, a una función que tenga un valor de retorno. Dicho valor simplemente se ignorará.

```
int f() { return 1; }

main() {
        print f();          // Válido
```

```
        f();            // Válido también (el valor de retorno se ignorará)
}
```

- No es necesario comprobar si existe una función *main*. Si no existe, es cierto que no podrá ejecutarse el programa, ya que el intérprete buscará este punto de entrada y, al no encontrarlo, indicará el error. Pero no es asunto del compilador (lo mismo que el compilador de Java no nos exige que todo fichero tenga un *main*).
- Tampoco es necesario comprobar que la función *main*, de haberla, sea la última del fichero. Puede aparecer en cualquier sitio. Sin embargo, si que es cierto que cualquier función definida debajo del *main* no podrá ser invocada. Pero si estas invocaciones no se hacen, el programa será válido.

Entregables en el examen

La práctica finalizada se entregará directamente el día del examen, no siendo necesaria una entrega previa.

Al examen, además de los fuentes de la práctica, habrá que llevar **obligatoriamente** los documentos con las especificaciones de las distintas fases del compilador en formato PDF. En concreto, las especificaciones que se piden son:

- Léxico del lenguaje expresado mediante Expresiones Regulares.
- Sintaxis del lenguaje mediante una Gramática Libre de Contexto.
- Descripción de los nodos del Árbol Abstracto (AST) mediante una Gramática Abstracta.
- Descripción de la fase de Comprobación de Tipos del análisis semántico mediante una Gramática Atribuida (no se pide la Gramática Atribuida de la etapa de Identificación).
- Descripción de la fase de Selección de Código mediante una Especificación de Código (no se pide la Gramática Atribuida de la etapa de Gestión de Memoria).

El no entregar dichos documentos o hacerlo de forma errónea o incompleta supondrá el no superar el examen práctico.

Raúl Izquierdo Castanedo

Profesor de Diseño de Lenguajes de Programación