Práctica P3

Intérprete para el lenguaje `P`

8 de mayo de 2019

Resumen

Esta es la tercera y última entrega práctica de la asignatura en régimen de evaluación continua. En ella se propone desarrollar un intérprete para un lenguaje, que llamaremos `P`, empleando las herramientas `LeX` y `Yacc` para la construcción del analizador léxico y sintáctico. Se propone una compilación en dos fases para poder soportar alguna de las características de `P`: en la primera fase, se construirá el árbol de sintaxis abstracta (`AST`) que representa los contenidos del documento (programa) de entrada; en la segunda y última fase, se recorrerá el `AST` ejecutando las acciones previstas en el código que representa.

1. Especificaciones del lenguaje

El lenguaje `P` se puede ver como un dialecto mínimo del lenguaje C. Comparte también características con el mini lenguaje presentado en la sesión de laboratorio impartida por el profesor Dušan Kolář. Se trata de un lenguaje no tipado, con estructura orientada a bloques y que contiene sentencias condicionales tanto de bifurcación como de iteración. Para dar soporte operativo al lenguaje, se incluyen varias funciones predefinidas básicas, tanto de tipo matemático como de entrada y salida.

1.1. Especificaciones léxicas

En esta subsección describimos los elementos léxicos que componen el lenguaje `P`. Cualquier carácter que no se corresponda con las reglas indicadas aquí deberá ser señalado como un error léxico, presentando el mensaje de error adecuado con indicación de la línea y columna del documento de entrada en que aparezca el carácter léxico incorrecto.

1.1.1. Identificadores y Palabras clave

Un identificador se compone de una secuencia no vacía de letras, números y '_' que comience por una letra o por '_'. Se distinguen letras mayúsculas y minúsculas. Todas las palabras clave (que irán apareciendo en el resto del texto) se deben escribir siempre en minúsculas.

1.1.2. Constantes numéricas

Las constantes numéricas se componen de una parte entera, opcionalmente seguida de una parte decimal separada por '.'. Podrán también representarse en notación exponencial. El exponente se compone de una letra 'e' o 'E' seguido de un signo '+' (opcional) o '-' y de una secuencia no vacía de números. Todas las constantes numéricas se almacenarán como números en coma flotante en doble precisión.

1.1.3. Constantes carácter y tiras de caracteres

Las constantes carácter y tiras de caracteres en `P` tendrán la misma representación que en el lenguaje `C`, aunque no está previsto incluir, en la implementación base, operaciones sobre estas entidades. Se contemplan exclusivamente para su uso como constantes en las funciones de entrada y salida.

1.1.4. Separadores

En P, los blancos, tabuladores, saltos de página, saltos de línea y retornos de carro se consideran caracteres de separación y serán ignorados por el analizador léxico. Las sentencias se podrán agrupar en bloques, delimitados por el carácter '{' y '}' (como en lenguaje C).

1.1.5. Comentarios

Los comentarios facilitan la documentación de programas escritos en P. Cualquier secuencia de caracteres comprendida entre la secuencia '/*' y '*/' será tratada como comentario (multilínea). Cualquier secuencia de caracteres comprendida entre la secuencia '//' y el salto de línea siguiente '"n' será considerada un comentario. No se admiten comentarios anidados. Los comentarios se incluyen a efectos puramente documentales, y serán ignorados por el analizador léxico.

1.1.6. Operadores

El lenguaje P contiene los operadores aritmético-lógicos más comunes y los paréntesis '(' y ')' para agrupar operaciones en un orden concreto dentro de expresiones complejas. La lista de operadores soportados es:

- **Operadores aritméticos**: '+' (suma), '-' (resta, cambio de signo), '*' (producto), '/' (división), '%' (división entera), '^' (potencia).
- **Operadores lógicos**: '>' (mayor), '<' (menor), '>=' (mayor o igual)', '<=' (menor o igual), '==' (igual), '!=' (diferente), '!' (not), '&&' (and), '||' (or).
- **Asignación**: '=' es el operador de asignación, que se podrá emplear en sentencias de asignación.

1.2. Especificaciones sintácticas

Un programa correcto en P se compone de una secuencia de una o más sentencias de los siguientes tipos:

- Asignación
- Sentencia vacía (NOP): ';'
- Sentencia bloque
- Sentencia condicional **if**
- Bucle condicional **while**
- Llamada a función de entrada/salida.

P no incluye sentencias de declaración de variables ni de tipos. El tipo de una variable viene asignado automáticamente por la primera asignación a dicha variable del resultado de una expresión. Tanto en las asignaciones como en alguna de las partes de otras sentencias se usarán **expresiones**, en las que se podrán combinar variables, constantes y llamadas a **funciones implícitas** del lenguaje P.

1.2.1. Asignaciones

Una asignación en P consta de un identificador (que no podrá ser una palabra reservada), seguido del operador de asignación **'='** y de una expresión, separados por cero o más caracteres de separación.

1.2.2. Sentencia bloque

Se puede agrupar una secuencia de cero o más sentencias en `P` encerrando la misma entre los caracteres '{' (inicio de bloque), '}' (fin de bloque). Los caracteres de inicio y fin de bloque podrán estar separados de los contenidos por cero o más caracteres separadores. Las sentencias de bloque se usan en `P` como componentes de sentencias compuestas.

1.2.3. Sentencia condicional if

Esta sentencia tiene una estructura y semántica semejante a la del lenguaje `C`. Consta de la palabra reservada **if** seguida de una expresión correcta en `P` encerrada entre paréntesis (que evaluará como condición de valor verdadero/falso), seguida de una sentencia bloque que se ejecutará cuando la condición se evalúe a 'verdadero' (algo diferente de cero). Opcionalmente, podrá contener un bloque introducido por la palabra clave **else** seguida de una sentencia bloque correcta que será la que se ejecute si la condición se evalúa a cero.

1.2.4. Bucle condicional while

Esta sentencia tiene una estructura y semántica semejante a la del lenguaje `C`. Consta de la palabra reservada **while** seguida de una expresión correcta en `P` encerrada entre paréntesis (que evaluará como condición de valor verdadero/falso), seguida de una sentencia bloque que se ejecutará mientras la condición se evalúe a 'verdadero' (algo diferente de cero).

1.2.5. Funciones de entrada/salida

El lenguaje `P` contiene funciones predefinidas (cuyos nombres no pueden usarse como identificadores) para facilitar entrada/salida de valores (por la entrada/salida estándar en la versión base). Estas funciones facilitan la presentación de resultados (salida) y la lectura de valores de variables proporcionados por el usuario (entrada). Las funciones soportadas en la implementación base de `P` son:

- **write**: La función **write** tomará uno o dos argumentos, separados por coma y encerrados entre paréntesis de apertura y cierre. Cuando se use la variante de dos argumentos, el primero será una constante de tipo carácter o tira de caracteres que se mostrará antes del valor de la expresión que se incluya en el segundo argumento. Cuando se use la variante de un argumento, se podrá usar como argumento una constante o variable, cuyo valor será mostrado en la salida estándar. Esta función no devuelve ningún valor.
- **read**: La función **read** tomará uno o dos argumentos, separados por coma y encerrados entre paréntesis de apertura y cierre. Cuando se use la variante de dos argumentos, el primero será una constante de tipo carácter o tira de caracteres que se mostrará como *prompt*. El segundo argumento en la variante de dos argumentos o el único argumento en la de uno será el nombre de una variable a la que se asignará el valor leído de la entrada estándar (dado que las variables en la implementación estándar son sólo numéricas, se transformará a tipo numérico la respuesta que escriba el usuario). Esta función no devuelve ningún valor.

1.2.6. Funciones predefinidas

El lenguaje `P` contiene también funciones predefinidas (cuyos nombres no pueden usarse como identificadores) para facilitar el cálculo de funciones matemáticas importantes. Las funciones predefinidas toman siempre un argumento y retornan un valor. El argumento se pasa a la función por valor. La implementación base de `P` deberá soportar las siguientes funciones predefinidas:

- **Funciones trigonométricas básicas**: **sin()**, **cos()**, **tan()**, **asin()**, **acos()**, **atan()**. Representan el cálculo del seno, coseno, tangente de un ángulo que se proporciona como argumento o del arco

cuyo seno, coseno o tangente se indique como argumento. Los ángulos se expresarán siempre en radianes.

- **Funciones matemáticas elementales**: **log()**, **log10()**, **exp()**, representando el logaritmo natural, el logaritmo decimal o el resultado de elevar **e** al valor proporcionado como argumento. **ceil()** para representar el entero más pequeño mayor o igual que el valor del argumento, **floor()** para representar el entero más grande menor o igual que el valor del argumento.

1.2.7. Expresiones

Las expresiones en lenguaje `P` combinan operaciones aritmético-lógicas de otras expresiones, o valores de constantes numéricas o resultados de llamadas a funciones que devuelven un valor escalar. La precedencia y asociatividad de los operadores aritméticos y lógicos es la estándar en lenguaje `C` y se podrá alterar mediante el uso explícito de paréntesis que fuercen una determinada interpretación o mejoren la lectura por el programador. Toda expresión se evalúa a un valor escalar que se podrá utilizar bien en otra expresión o en los lugares indicados de las sentencias del lenguaje `P`.

2. Extensiones

A las especificaciones base contenidas en las secciones anteriores se podrán añadir alguna de las extensiones que se indican a continuación. Será preciso incorporar al menos una para poder aspirar a la máxima calificación. La especificación concreta de estas extensiones correrá a cuenta del usuario y su adecuación/complejidad será tenida en cuenta en la calificación de la misma.

- **Variables y operaciones con caracteres**: Se propone incorporar caracteres y tiras de caracteres como operandos o resultados de expresiones, así como alguna función básica de edición de las mismas.
- **Funciones declaradas por el usuario**: Se propone incorporar una sentencia que permita declarar funciones de varios argumentos (siempre pasados por valor) que devuelven siempre un escalar (o nada). Se deberá comprobar, a través de la tabla de símbolos, que la definición y el uso son compatibles. Para definir la función, se empleará la palabra reservada **def** en sustitución del tipo devuelto que aparecería en C y los argumentos no contendrán declaraciones de tipos.
- **Bucle for**: Se propone incorporar un bucle con la sintaxis del **for** del lenguaje C.

3. Implementación

El intérprete de `P` se construirá a partir de la especificación `LeX`, `Yacc` y de los ficheros `.c, .h` necesarios. El resultado será un programa ejecutable `pcc` que admitirá como argumento el nombre de un fichero de entrada en el que estará el programa en `P` que hay que interpretar/ejecutar.

Se señalarán los errores léxicos y sintácticos indicando, al menos, la línea del fichero de entrada en que aparecen (aunque este valor pueda ser, en general, aproximado). Cualquier error léxico o sintáctico supondrá la parada inmediata del intérprete.

A partir del programa fuente en `P`, el intérprete construirá un `AST` y mantendrá una **tabla de símbolos** a la que se hará referencia para acceder a las variables, funciones y constante. La implementación de ambos componentes puede partir de la propuesta en la sesión de laboratorio del profesor Dušan Kolář. Para la implementación del intérprete para las especificaciones base puede servir directamente. Para alguna de las extensiones habrá que modificarla.

La interpretación del programa se realizará en un segundo paso tras el análisis del mismo, recorriedo el `AST` generado a partir del programa de entrada.

4. Ejemplos

A título ilustrativo, las figuras 1 y 2 muestran un par de programas escritos en `P` que deberían ser interpretados correctamente por el `pcc` que construya. Se presentan a título ilustratuivo y se espera que cada estudiante construya sus propios ejemplos correctos e incorrectos para probar el intérprete.

```
/*
 * Programa sencillo que muestra una serie de números
 */
write("### Muestra los elementos de una progresión aritmética ###\n") ;
read ("_Primer elemento .......: ", a_0) ;
write("_Razón ..................: ") ; read(razon) ;
read ("_Número de elementos ...: ", nelem) ;

n_i = 0 ; a_i=a_0 ; // Inicial
while (n_i < nelem) {
    write ("___Elemento (", n_i) ; write ("):  ", a_i) ; write('\n') ;
    // Siguiente elemento
    a_i = a_i + razon ;
    n_i = n_i + 1  ;
}
```

Figura 1: Un programa simple en `P`.

5. Normas de entrega

Todos los ficheros y directorios que se entregan deberán estar contenidos en un directorio que tenga como nombre su NIF (letra final en mayúscula) (e.g. 12345678X). Dentro de ese directorio base deberán aparecer:

- **pcc.l**: Fichero que contiene la especificación `LeX` del analizador léxico.
- **pcc.y**: Fichero que contiene la especificación `Yacc` (o `bison`) del analizador sintáctico.
- **.c, .h**: Ficheros de cabecera (.h) o fuentes en C (.h) que considere necesarios para dar soporte al intérprete construido.
- **makefile**: Fichero para la orden `make` o script en `bash` para generar el intérprete.
- **test**: Directorio que contendrá ejemplos ilustrativos de programas en `P` correctos e incorrectos para poder comprobar el funcionamiento del intérprete construido.
- **doc**: Directorio que contendrá el informe escrito (los fuentes y la versión PDF resultante) con el que documente el trabajo realizado.

El intérprete debe funcionar bajo sistema operativo Linux. Cualquier biblioteca que se emplee deberá ser incluida en las referencias y justificada en el informe.

Todos los contenidos del directorio base se comprimirán en un fichero NIF.zip (e.g. 12345678X.zip) que será subido como entrega de la actividad P3 en la página de la asignatura en campusvirtual.uva.es en las fechas que se indican en la tarea en el Moodle.

6. Evaluación

Este supuesto representa un 10 % de la calificación final de la asignatura (1 punto sobre 10). El 70 % de la calificación de esta entrega se asociará a la elaboración de la solución y el 30 % restante a la documentación de la misma. El profesor podrá requerir a cualquier estudiante aclaraciones sobre la entrega realizada antes de emitir la calificación, si lo estima conveniente. Cualquier evidencia de plagio

supondrá la obtención de una calificación de '0' puntos en esta entrega. A la hora de calificar se tendrán en cuenta los siguientes criterios:

- **Funcionamiento del intérprete**. El funcionamiento incorrecto total o parcial será penalizado.
- **Nivel de implementación**. El intérprete que se ajuste a los requisitos básicos especificados en este documento podrá obtener una calificación máxima del 80 % de la parte asociada a la solución. Para poder obtener el 100 % se deberá incluir al menos una de las extensiones indicadas como posibles en la sección 2.
- **Documentación**: Se valorará con hasta un 30 % de la calificación final la calidad de la documentación presentada, teniendo en cuenta, al menos, los siguientes aspectos: orden, claridad, precisión, justificación de las decisiones de diseño/implementación tomadas, referencias empleadas.

```
/*
 * Muestra la sucesión de puntos de un arco de circunferencia
 * centrada en el origen y de radio R, comprendidos entre un
 * ángulo inicial (en fracciones de pi) y uno final.
 */
write("### Muestra los puntos de un arco de circunferencia ###\n") ;
write("_Angulo inicial [unidades de PI] ... : ") ; read(aini) ;
read ("_Angulo final [unidades de PI] ..... : ", afin) ;
read ("_Radio ..... : ", radio) ;
read ("_Numero .... : ", NumPtos) ;

// Obtenemos el valor de PI
PI=acos(-1);

// Cálculo y normalización a circunferencia del incremento de ángulos
// No es la mejor implementación, claro ...
adelta = (afin - aini) / NumPtos ;
if (adelta > 2) {
    while (adelta > 2) {
        adelta = adelta - 2 ;
    }
}
adelta = adelta * PI ;

// Valores iniciales
n_pto = 0 ;
a_pto = aini*PI ;
x_pto = radio * cos(a_pto) ;
y_pto = radio * sin(a_pto) ;

/* Mientras no alcancemos el máximo de puntos a mostrar */
while (n_pto < NumPtos) {
    write ("___Punto (", n_pto) ;
    write ("), ángulo (", a_pto) ; write ("): [", x_pto); write (", ", y_pto);
    write("]\n") ;
    // Siguiente elemento
    a_pto = a_pto + adelta ;
    x_pto = radio * cos(a_pto) ;
    y_pto = radio * sin(a_pto) ;
    n_pto = n_pto + 1 ;
}
```

Figura 2: Algo de matemáticas en P.