Prácticas

Contenido de la página:

Normas generales:

La práctica se podrá realizar en grupos de, como máximo, 3 personas. La puntuación obtenida no depende del número de integrantes del grupo, tampoco tiene por qué ser igual para todos los integrantes.

No se permite la integración de personas en un grupo después de la primera entrega.

Sí se permite la salida de personas de un grupo, dejando claro en una entrevista con el profesor, quién continúa con la práctica y quién no hace la práctica o decide empezar una nueva.

Material de entrega:

- Una **memoria escrita** en formato electrónico que incluya:
  - Una descripción del trabajo realizado, así como cualquier anotación o característica que se desee hacer notar, **sin incluir listados fuente**.
  - 8 casos de prueba de los cuales, 4 han de ser correctos y 4 erróneos, de forma que permitan observar el comportamiento del compilador.
  - Recordad: ***LO BUENO, SI BREVE DOS VECES BUENO***
- **Aplicación informática** que implemente la funcionalidad requerida para la entrega correspondiente (léxico, sintáctico o completa):
  - La aplicación se debe poder ejecutar sobre ***plataforma Windows*** 7.
  - Los listados fuente de las especificaciones e implementación.
  - Los ficheros asociados a los casos de prueba que aparecen en la memoria.
  - La calidad del material entregado es responsabilidad de los estudiantes. En caso de encontrar una entrega con virus, o defectuosa, dicha entrega se considerará suspensa.

Calificación:

La calificación de la práctica se divide en tres niveles:

- **aprobado** (hasta 5), completando la parte obligatoria.
- **notable** (hasta 7), alcanzando el grado de aprobado y proporcionando notificación de errores detallada y completando 1 tipo de datos y dos sentencias de control de flujo de la parte opcional.
- **sobresaliente** (hasta 9), alcanzando el grado de notable y proporcionando recuperación de errores así como toda la parte opcional.

Además se otrogará un punto extra en función de la calidad de la memoria final.

Plazos de entrega:

- **Marzo 2017** .Evaluación ordinaria.
- **Mayo 2017.** Evaluación ordinaria.
- **Junio-Julio 2017.** Evaluación extraordinaria.

Comienzo de la página

Especificación de la práctica:

La práctica consiste en el **diseño e implementación de un visualizar de código** para el subconjunto de un lenguaje de programación similar a C. Se permite utilizar herramientas de generación automática estilo Lex/Yacc.

Parte obligatoria:

Especificaciones léxicas

Los elementos del lenguaje que aparecen entrecomillados en la gramática, deben aparecer **tal cual** (sin las dobles comillas) en cualquier programa correctamente escrito en este lenguaje, el resto de elementos se especifican a continuación.

Los **identificadores** son ristras de símbolos compuestas por letras, dígitos (de base decimal), símbolos "$" y guiones bajos "_" (underscore). Empiezan obligatoriamente por una letra o el símbolo "$". Ejemplos correctos: `contador` , `contador1` , `$acumulador_total_2`

Las **constantes numéricas** pueden ser de **dos tipos**, representados en la gramática por los símbolos terminales **`constint`** y **`constfloat`**, y se pueden especificar en **tres bases distintas.**

- Bases posibles: decimal, octal y hexadecimal.
    - Las constantes numéricas en **base decimal** (10), estarán compuestas por los dígitos decimales, del 0 al 9, además de los símbolos necesarios en función del tipo al que pertenezcan (enteras o reales).
    - Las constantes numéricas en **base octal** (8), **siempre comienzan con el dígito** "**0**" y estarán compuestas por los dígitos octales, del 0 al 7, además de los símbolos necesarios en función del tipo al que pertenezcan (enteras o reales).
    - Las constantes numéricas en **base hexadecimal** (16), **siempre comienzan con** "**0x**" y estarán compuestas por los dígitos decimales, del 0 al 9 y las letras de la A a la F, además de los símbolos necesarios en función del tipo al que pertenezcan (enteras o reales).
- Tipos posibles: enteros y reales.
    - Las constantes numéricas **enteras** son una ristra de dígitos, opcionalmente precedidas de un signo + o -.
    - Las constantes numéricas **reales** son dos ristras de dígitos, opcionalmente precedidas de un signo + o - y separadas por el punto decimal.
- Ejemplos de constantes correctamente escritas:
    - Enteras:
        - Base decimal: `+123` , `-69` , `45`
        - Base octal: `0+123` , `0-64` , `045`
        - Base hexadecimal: `0x+123` , `0x-A6F9` , `0xFFFF`
    - Reales:
        - Base decimal: `+123.456` , `-0.69` , `45.0`
        - Base octal: `0+123.456` , `0-64.77` , `045.16`, 00.35
        - Base hexadecimal: `0x+123.0` , `0x-E.A6F9` , `0x0.FFFF`

Las **constantes literales**, representadas en la gramática por el símbolo terminal **`constlit`**, son ristras de símbolos entre comillas simples,`'contenido de la constante literal'`**.** El contenido de las constantes puede ser cualquier carácter que pueda aparecer en el programa fuente. Si se desea que aparezca la comilla simple como contenido, ésta debe ir precedida del símbolo ' ****', por ejemplo: el contenido de la constante **`'constante literal con \'contenido\' entrecomillado'`** sería: **`constante literal con 'contenido' entrecomillado`**

El formato de los **comentarios de propósito general** es: cualquier carácter que pueda aparecer en el código fuente, entre los símbolos ' **`//`**' hasta el final de la línea, o entre las parejas de símbolos `/*` y `*/`. Lógicamente el contenido

del comentario no puede tener los caracteres de finalización del mismo. Este tipo de comentarios pueden aparecer antes o después de cualquier elemento del leguaje, pero nunca dentro.

Sintaxis general

Un programa está compuesto por sucesivas declaraciones de funciones o procedimientos. La diferencia entre ambos es que los procedimientos devuelven el tipo void.

```
PROGRAM ::= PART PROGRAM | PART
PART ::= TYPE RESTPART
RESTPART ::= ident "(" LISTPARAM ")" BLQ
BLQ ::= "{" SENTLIST "}"
LISTPARAM ::= LISTPARAM "," TYPE ident | TYPE ident
TYPE ::= "void" | "int" | "float"
```

dentro de las funciones y procedimientos se pueden encontrar senctencias de declaración de variables, asignación y llamadas a funciones y procedimientos.

```
SENTLIST ::= SENTLIST SENT | SENT
SENT ::= TYPE LID ";" | ident "=" EXP ";" | ident "(" LEXP ")" ";" |
"return" EXP ";"
LID ::= ident | LID "," ident
LEXP ::= EXP | LEXP "," EXP
EXP ::= EXP OP EXP | FACTOR
OP ::= "+" | "-" | "*" | "/" | "%"
FACTOR ::= ident "(" LEXP ")" | "(" EXP ")" | ident | constint |
constfloat
  | constlit
```

Parte opcional:

Sentencias de control de flujo de ejecución:

```
SENT ::= if "(" LCOND ")" "then" BLQ "else" BLQ
        | "for" "(" ident ASIG EXP ";" LCOND ";" ident ASIG EXP ")" BLQ
        | "while" "(" LCOND ")" BLQ
        | "do" BLQ "until" "(" LCOND ")"
        | BLQ

LCOND ::= LCOND OPL LCOND | COND | "not" COND
OPL ::= "or" | "and"
COND ::= EXP OPR EXP
OPR::= "==" | "<" | ">" | ">=" | "<="
```

Tipos de datos:

Estructuras:

```
PART ::= ... | "struct" ident "{" LFIELD "}" ";"
LFIELD ::= LFIELD TYPE LID ";"  | TYPE LID ";"

ID ::= ident | ident "." ident
```

```
FACTOR ::= ident "(" LEXP ")" | "(" EXP ")" | ID | constint |
constfloat
 | constlit
```

Matrices:

```
LID ::= ID | LID "," ID
ID ::= ident | ident LDIM
LDIM ::= LDIM "[" constint "]" | "[" constint "]"
FACTOR ::= ident "(" LEXP ")" | "(" EXP ")" | ID | constint |
constfloat
 | constlit
```

Comienzo de la página

última modificación: 31 Enero, 2017