UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior

Procesadores de lenguaje

Caso práctico

Desarrolle un traductor predictivo, recursivo y descendente para un subconjunto de instrucciones de un lenguaje similar a C. El traductor debe generar código intermedio para una máquina abstracta de pila.

Dados los componentes léxicos, los lexemas del lenguaje y su gramática, desarrolle un traductor que cumpla los siguientes requisitos:

- El analizador léxico debe leer los programas de prueba almacenados en ficheros de texto. Debe descartar los espacios, tabuladores y saltos de línea del fichero de entrada.
- El analizador léxico debe reconocer las palabras reservadas, los operadores y los caracteres delimitadores del lenguaje. El fichero `lexemas.txt` define las parejas de componentes léxicos y de lexemas del lenguaje.
- El analizador léxico debe reconocer como identificadores los lexemas que empiecen por una letra seguida de cero o más letras, dígitos o guiones bajos, siempre que el lexema no sea una palabra reservada.
- El analizador léxico debe reconocer las secuencias de dígitos como números enteros o números reales.
- El analizador léxico debe descartar los comentarios del programa. Un comentario de una línea comienza por los caracteres //. Para delimitar un comentario de una o más líneas se usan las secuencias de caracteres /* y */. No se admiten comentarios anidados, cuando se encuentra el delimitador de inicio de un comentario se ignora el resto de caracteres hasta encontrar el delimitador de fin.

- El analizador sintáctico debe verificar que los programas de prueba cumplen la gramática del lenguaje y su semántica.
- El analizador sintáctico debe indicar si el proceso de compilación ha sido correcto o si se han encontrado errores sintácticos o semánticos. En caso de que se encuentre un error, debe mostrar una breve descripción del error y la línea del programa en la que se ha producido.
- El analizador sintáctico debe comprobar que todos los identificadores del programa han sido declarados una sola vez y que se cumplen las reglas de verificación de tipos de datos para las instrucciones de asignación y para las expresiones del lenguaje.
- El analizador sintáctico debe generar un fichero de texto con el código intermedio para una máquina abstracta de pila. La generación de código solo se realiza para los programas que usan tipos de datos primitivos.

Componentes léxicos y lexemas

```
// Operadores relacionales y de asignación, aritméticos y lógicos

greater_than                    >
greater_equals                  >=
less_than                       <
less_equals                     <=
equals                          ==
not_equals                      !=
assignment                      =

add                             +
subtract                        -
multiply                        *
divide                          /
remainder                       %

and                             &&
or                              ||
not                             !

// Delimitadores

dot                             .
comma                           ,
semicolon                       ;
open_parenthesis                (
closed_parenthesis              )
open_square_bracket             [
closed_square_bracket           ]
open_bracket                    {
closed_bracket                  }
end_program                     #

// Palabras reservadas

boolean                         boolean
do                              do
else                            else
false                           false
float                           float
if                              if
int                             int
main                            main
print                           print
while                           while
true                            true
void                            void
```

Gramática

programa	→	**void main {** *declaraciones instrucciones* **}**
declaraciones	→	*declaración-variable declaraciones* | **ε**
declaración-variable	→	*tipo-vector* **id ;** |
		tipo-primitivo lista-identificadores **;**
tipo-primitivo	→	**int** | **float** | **boolean**
tipo-vector	→	**int [num]** | **float [num]** | **boolean [num]**
lista-identificadores	→	**id** *asignacion-declaración más-identificadores*
más-identificadores	→	**,** **id** *asignacion-declaración más-identificadores* | **ε**
asignacion-declaración	→	**=** *expresión-logica* | **ε**
instrucciones	→	*instrucción instrucciones* | **ε**
instrucción	→	*declaración-variable* |
		variable **=** *expresión-logica* **;** |
		if (*expresión-lógica***)** *instrucción* |
		if (*expresión-lógica***)** *instrucción* **else** *instrucción* |
		while (*expresión-lógica***)** *instrucción* |
		do *instrucción* **while (***expresión-lógica***) ;** |
		print (*variable***) ;** |
		{ *instrucciones* **}**
variable	→	**id** | **id [***expresión***]**
expresión-lógica	→	*expresión-lógica* **||** *término-lógico* |
		término-lógico
término-lógico	→	*término-lógico* **&&** *factor-lógico* |
		factor-lógico
factor-lógico	→	**!** *factor-lógico* | **true** | **false** |
		expresión-relacional
expresión-relacional	→	*expresión operador-relacional expresión* |
		expresión
operador-relacional	→	**<** | **<=** | **>** | **>=** | **==** | **!=**
expresión	→	*expresión* **+** *término* |
		expresión **-** *término* |
		término
término	→	*término* ***** *factor* |
		término **/** *factor* |
		término **%** *factor* |
		factor
factor	→	**(***expresión***)** | *variable* | **num**

Programas de prueba para el análisis sintáctico y semántico

Programa 1

```
void main {
   int [10] f;

   // sucesión de Fibonacci

   f[0] = 0;
   f[1] = 1;

   int i = 2;

   while (i < 10) {
      f[i] = f[i-1] + f[i-2];

      i = i + 1;
   }

   i = 0;

   while (i < 10) {
      print(f[i]);

      i = i + 1;
   }
}
```

Salida por la consola

```
Programa compilado correctamente

Tabla de símbolos
<'i', int>
<'f', array (int, 10)>
```

Programa 2

```
void main {
   int [10] f;

   f[0] = 0;

   int i = 1;

   do {
      if (i % 2 == 0)
         f[i] = f[i-1] + i;
      else
         f[i] = f[i-1] + 2*i;

      i = i + 1;
   } while (i < 10);

   print(f[9]);
}
```

Salida por la consola

```
Programa compilado correctamente

Tabla de símbolos
<'i', int>
<'f', array (int, 10)>
```

Programa 3

```
void main {
   int a = 1, b = 2, c = (25 * (2 + a)) / (2 * b), d = a + 2*b + c;

   if (d >= 10 && d <= 20 || d < 5) {
      c = d - 5;
   } else {
      c = d + 5;
   }

   int i = 0;

   while (i <=10 && c >= 0) {
      c = c - 1;
      i = i + 1;
   }

   if (a <= 10)
      c = 1;
   else {
      c = 2;

      do {
         c = c + 2;
         a = a - 1;
      } while (a >= 0);
   }

   print(a);
   print(b);
   print(c);
   print(d);
}
```

Salida por la consola

```
Programa compilado correctamente

Tabla de símbolos
<'a', int>
<'b', int>
<'c', int>
<'d', int>
<'i', int>
```

Programa 4

```
void main {
   int a, b, c, d;
   float x = 0.0;

   a = 1;
   b = 2;
   c = (25 * (2 + a)) / (2 * b);
   d = a + 2*b + c;

   x = a + b + c + d;
}
```

Salida por la consola

```
Error en la línea 10, incompatibilidad de tipos en la instrucción de asignación

Tabla de símbolos
<'a', int>
<'b', int>
<'c', int>
<'d', int>
<'x', float>
```

Programa 5

```
void main {
   int a, b, c;

   a = 1;
   b = 2;
   c = (25 * (2 + a)) / (2 * b);

   int d = a + 2*b + c;

   e = d * (2 + 3 * a);
}
```

Salida por la consola

```
Error en la línea 10, identificador 'e' no declarado

Tabla de símbolos
<'a', int>
<'b', int>
<'c', int>
<'d', int>
```

Programa 6

```
void main {
   int a, b, c, d;
   int d = 2;

   /*
    * este programa declara la variable 'd' dos veces
    * la variable 'x' no se declara
    */

   a = 1;
   b = 2;
   c = (25 * (2 + a)) / (2 * b);
   d = a + 2*b + c;

   x = a + b + c + d;
}
```

Salida por la consola

```
Error en la línea 3, identificador 'd' ya declarado
Error en la línea 15, identificador 'x' no declarado

Tabla de símbolos
<'a', int>
<'b', int>
<'c', int>
<'d', int>
```

Programas de prueba para la generación de código

Programa 7

```
void main {
   int y = 2, m = 1, d = 3;

   int x = (14 * y) / 4 + (16 * m + 4) / 2 + d;
}
```

Código de la máquina de pila

```
lvalue y
push 2
=
lvalue m
push 1
=
lvalue d
push 3
=
lvalue x
push 14
rvalue y
*
push 4
/
push 16
rvalue m
*
push 4
+
push 2
/
+
rvalue d
+
=
halt
```

Programa 8

```
void main {
   int f = 1;
   int i = 1;

   while (i <=5) {
      f = f * i;
      i = i + 1;
   }

   print(f);
}
```

Código de la máquina de pila

```
lvalue f
push 1
=
lvalue i
push 1
=
label_0:
rvalue i
push 5
<=
gofalse label_1
lvalue f
rvalue f
rvalue i
*
=
lvalue i
rvalue i
push 1
+
=
goto label_0
label_1:
print f
halt
```

Programa 9

```
void main {
   int f = 1;
   int i = 1;

   do {
      f = f * i;
      i = i + 1;
   } while (i <=5);

   print(f);
}
```

Código de la máquina de pila

```
lvalue f
push 1
=
lvalue i
push 1
=
label_0:
lvalue f
rvalue f
rvalue i
*
=
lvalue i
rvalue i
push 1
+
=
rvalue i
push 5
<=
gofalse label_1
goto label_0
label_1:
print f
halt
```

Programa 10

```
void main {
   int i = 2, f_0 = 0, f_1 = 1, f_2;

   while (i <= 10) {
      f_2 = f_1 + f_0;
      f_0 = f_1;
      f_1 = f_2;
      i = i + 1;
   }

   print(f_2);
}
```

Código de la máquina de pila

```
lvalue i
push 2
=
lvalue f_0
push 0
=
lvalue f_1
push 1
=
label_0:
rvalue i
push 10
<=
gofalse label_1
lvalue f_2
rvalue f_1
rvalue f_0
+
=
lvalue f_0
rvalue f_1
=
lvalue f_1
rvalue f_2
=
lvalue i
rvalue i
push 1
+
=
goto label_0
label_1:
print f_2
halt
```

Programa 11

```
void main {
   int i = 10;

   if (i % 2 == 0)
      i = i + 2;
   else
      i = i - 2;

   print(i);
}
```

Código de la máquina de pila

```
lvalue i
push 10
=
rvalue i
push 2
%
push 0
==
gofalse label_0
lvalue i
rvalue i
push 2
+
=
goto label_1
label_0:
lvalue i
rvalue i
push 2
-
=
label_1:
print i
halt
```

Programa 12

```
void main {
   int si = 0, sp = 0;

   int i = 1;

   while (i <= 10) {
      if (i % 2 == 0) {
         si = si + i;
      } else {
         sp = sp + i;
      }

      i = i + 1;
   }

   print(si);
   print(sp);
}
```

Código de la máquina de pila

```
lvalue si
push 0
=
lvalue sp
push 0
=
lvalue i
push 1
=
label_0:
rvalue i
push 10
<=
gofalse label_1
rvalue i
push 2
%
push 0
==
gofalse label_2
lvalue si
rvalue si
rvalue i
+
=
goto label_3
label_2:
lvalue sp
rvalue sp
rvalue i
+
=
label_3:
lvalue i
rvalue i
push 1
+
=
goto label_0
label_1:
print si
print sp
halt
```

El repertorio de instrucciones de la máquina de pila

Una máquina de pila utiliza memorias separadas para instrucciones y datos. Las operaciones aritméticas se realizan con los valores almacenados en la pila.

La siguiente tabla detalla el repertorio de instrucciones de la máquina de pila.

Instrucción	Descripción
push *value*	Apila *value*
rvalue *l*	Apila el valor almacenado en la localidad de memoria con etiqueta *l*
lvalue *l*	Apila la dirección almacenada en la localidad de memoria con etiqueta *l*
pop	Elimina el valor almacenado en el tope de la pila
=	El rvalue del tope de la pila se almacena en la dirección del lvalue inmediato anterior y ambos se eliminan de la pila
operador binario	Los operadores aritméticos binarios +, -, *, /, % se aplican a los dos últimos valores almacenados en la pila. Después de eliminar ambos operandos de la pila, se almacena el resultado de la operación en la pila De forma similar a los operadores aritméticos, los operadores lógicos y los operadores relacionales >, <, >=, <=, ==, != se aplican también a los dos últimos valores almacenados en la pila
label *label*	Etiqueta una instrucción con *label*
goto	Salto incondicional a la etiqueta *label*
gofalse *label*	Elimina el tope de la pila, si el valor es cero salta a la etiqueta *label*
gotrue *label*	Elimina el tope de la pila, si el valor es distinto de cero salta a la etiqueta *label*
halt	Finaliza la ejecución del programa

Fecha de entrega

La entrega del caso práctico es requisito indispensable para tener derecho a presentar el examen final ordinario. El caso práctico se realiza en grupos de dos alumnos.

Cada grupo debe presentar su trabajo antes del examen ordinario y enviar por correo electrónico un fichero ZIP con el código fuente.

Evaluación

La nota mínima para aprobar el caso práctico es 5,0. En caso de que durante la defensa del proyecto alguno de los alumnos no responda correctamente a las cuestiones planteadas, el caso práctico se considera suspenso para ambos alumnos.

El criterio de evaluación es el siguiente:

- [5,0 puntos] Análisis sintáctico de los programas de prueba
- [1,0 punto] Declaración de variables y de la tabla de símbolos
- [2,0 puntos] Verificación de tipos de datos
- [2,0 puntos] Generación de código para una máquina de pila