Traducción de miniC

Realizado por: Grupo 2.2, Convocatoria Junio

Daniel Ibáñez Molero DNI: 48658416F daniel.ibanez1@um.es
Ginés Miguel Cánovas Méndez DNI: 23333196H ginesmiguel.canovasm@um.es

Explicación de la práctica:

La practica consiste en la implementación funcional del lenguaje MiniC y su compilación consistente en la traducción de sus órdenes a ensamblador para poder ser ejecutado. Consiste en tres apartados principales: Análisis del léxico, sintáctico y semántico.

Los archivos principales en los que desarrollamos nuestro compilador se llaman comp.l y comp.y.

En comp.l declaramos las palabras clave que nuestro lenguaje va a reconocer, así como ciertas restricciones de estas (como el tamaño máximo permitido de nuestras cadenas). Este archivo será compilado con Flex, y generará un 'token' para todas nuestras palabras clave, que será analizado por calc.y.

En calc.y primero incorporaremos la funcionalidad de reconocer errores en la declaración de variables y constantes, como la redeclaración de estas, redefinición de constantes, etc. También procuraremos el funcionamiento adecuado de nuestras cadenas. Para hacer ambas cosas, nos serviremos de una estructura de lista enlazada llamada Lista l que nos servirá para almacenar ambas.

Finalmente, en calc.y indicaremos el código mips que se ha de generar dados los tokens recibidos, los cuales se almacenan en una estructura de lista enlazada que llamamos ListaC codigo. Para cada secuencia de tokens reconocida, insertaremos en esta lista las operaciones MIPS equivalentes, haciendo uso de los registros en los que vienen los operandos y procurando liberar estos cuando ya hayan sido usados. Hacemos uso de la línea %left para indicar la prioridad de nuestros operandos. Este archivo será compilado con Bison.

Manual de MiniC:

Nuestro lenguaje permite el uso de las siguientes expresiones:

- `funcion *1* comienzo *2* *3* fin`

Forma de inicializar nuestro lenguaje. *1* Será el nombre que le queramos dar, *2* serán las declaraciones de constantes, variables etc. y *3* las palabras clave que queramos usar con esas.

- `variables *1*.`
- `constantes *1*.`
- `variables *1*,*2*,*etc*.`

Forma de declarar variables y constantes. Se pueden declarar varias en la misma línea de la forma encontrada arriba. *1*, *2*… serán sus identificadores.

- `*1* = *x*`
- `*1*,*2* = *x*`

Forma de asignar valores a las variables y constantes. Las constantes no pueden ser redefinidas. *1*… serán sus identificadores y *x* su valor asignado.

- `si *1* entonces *2*`
- `si *1* entonces *2* sino *3*`
- `si *1* entonces comienzo *…* fin`

Condicionales lógicos. El valor 0 se considera verdadero y el resto falso, por lo que si *1* es igual a cero se hará *2*. En caso de no serlo y de usar la segunda estructura, se realizará *3*.

- `mientras *1* hacer *2*`
- `mientras *1* hacer comienzo *…* fin`

Bucle. Realiza una acción mientras que se cumpla *1*.

- `imprimir *1*.`
- `leer *1*.`

Imprime por pantalla y lee *1*, respectivamente.

Operaciones aritméticas:

- `*1* + *2*`
- `*1* - *2*`
- `*1* * *2*`
- `*1* / *2*`
- `-*1*`
- `(*1*+*2*)* *3*`

Estas son las operaciones matemáticas de las que es capaz nuestro lenguaje. Siempre serán realizadas con enteros. Se puede aplicar precedencia entre ellas con el uso de paréntesis, como se muestra en la ultima expresión. El resultado de la operación puede ser asignado a una variable, impreso por pantalla, analizado en un bucle, etc.

Ejemplos:

Entrada:

```
funcion prueba
comienzo
    constantes a=0, b=0.
    variables c=5+2-2.
    imprimir "Inicio del programa\n".
    si a entonces imprimir "a","\n".
    sino si b entonces imprimir "No a y b\n".
        sino mientras c hacer
            comienzo
                imprimir "c = ",c,"\n".
                c = c-2+1.
            fin
    imprimir "Final","\n".
fin
```

Salida:

```
.data
_a: .word 0
_b: .word 0
_c: .word 0
$str1: .asciiz "Inicio del programa\n"
$str2: .asciiz "a"
$str3: .asciiz "\n"
$str4: .asciiz "No a y b\n"
$str5: .asciiz "c = "
$str6: .asciiz "\n"
$str7: .asciiz "Final"
$str8: .asciiz "\n"
```

```
.text
.globl main
main:
li $t0,0
sw $t0,_a
li $t0,0
sw $t0,_b
li $t0,5
li $t1,2
add $t2,$t0,$t1
li $t0,2
sub $t1,$t2,$t0
sw $t1,_c
li $v0,4
la $a0,$str1
syscall
lw $t0,_a
beqz $t0,$etiq4
li $v0,4
la $a0,$str2
syscall
li $v0,4
la $a0,$str3
syscall
b $etiq5
$etiq4:
lw $t1,_b
beqz $t1,$etiq2
li $v0,4
la $a0,$str4
syscall
b $etiq3
$etiq2:
$etiq0:
lw $t2,_c
beqz $t2,$etiq1
li $v0,4
la $a0,$str5
syscall
lw $t3,_c
li $v0,1
move $a0,$t3
syscall
li $v0,4
la $a0,$str6
syscall
lw $t3,_c
li $t4,2
sub $t5,$t3,$t4
li $t3,1
```

```
add $t4,$t5,$t3
sw $t4,_c
b $etiq0
$etiq1:
$etiq3:
$etiq5:
li $v0,4
la $a0,$str7
syscall
li $v0,4
la $a0,$str8
syscall
li $v0,10
syscall
```