

Generación de código

Sentencias y declaraciones

Actualización de sintactico.y

```
%union {  
    char *str;  
    ListaC codigo;  
}
```

```
%type <codigo> expression statement statement_list print_item  
                    print_list read_list
```

Generación de código de asignaciones (1)

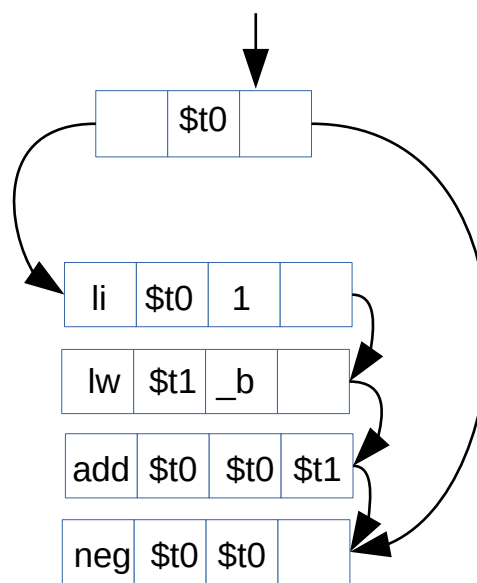
```
test1() {  
    var int a;  
    const int b = 3;  
    a = -(1 + b);  
}
```

registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

statement \rightarrow id = expression ; { ?? }

\$ $\$$ \$1 \rightarrow a\$ \$3



statement

id = expression ;

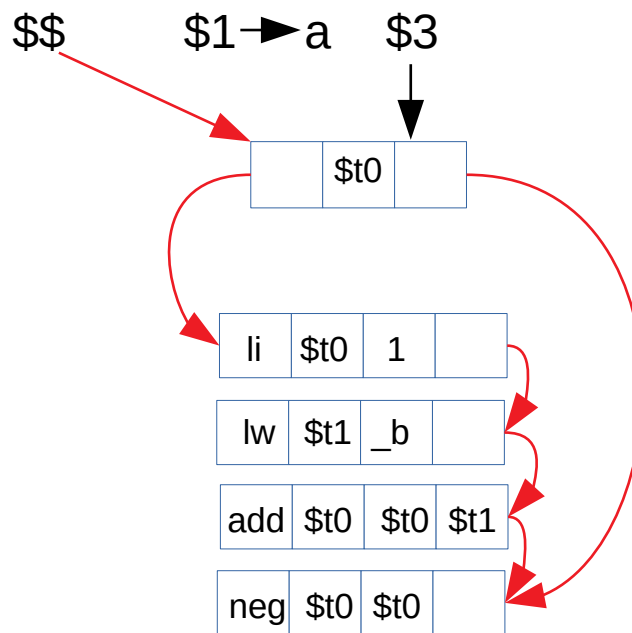
Generación de código de asignaciones (2)

```
test1() {  
    var int a;  
    const int b = 3;  
  
    a = -(1 + b);  
  
}
```

registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

statement \rightarrow id = expression ; { ?? }



statement

id = expression ;

Generación de código de asignaciones (3)

```
test1() {
```

```
    var int a;  
    const int b = 3;
```

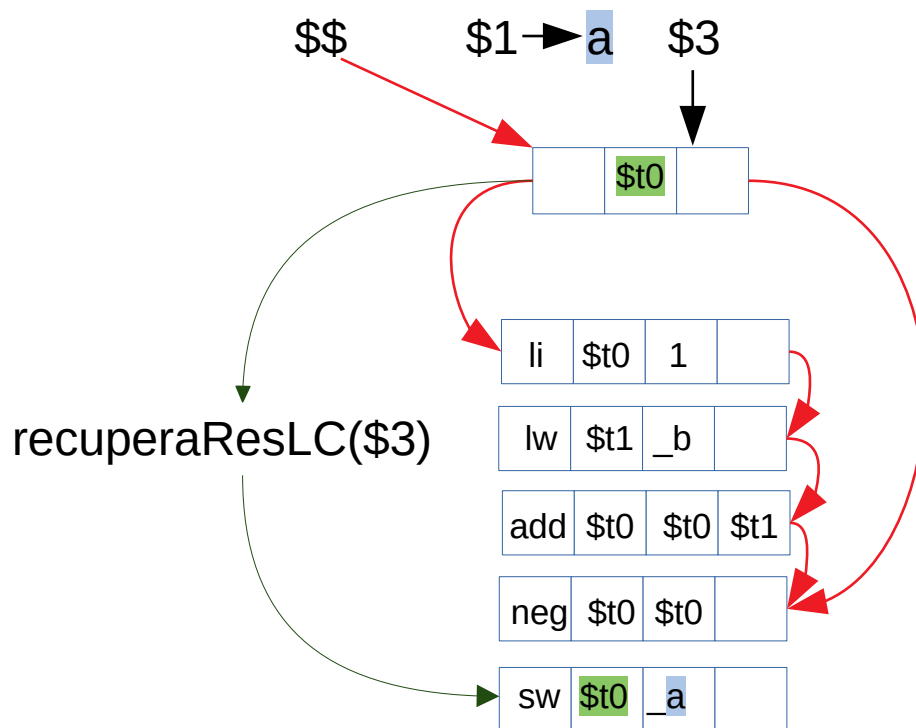
```
    a = -(1 + b);
```

```
}
```

registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

statement \rightarrow id = expression ; { ?? }



statement

id = expression ;

Generación de código de asignaciones (4)

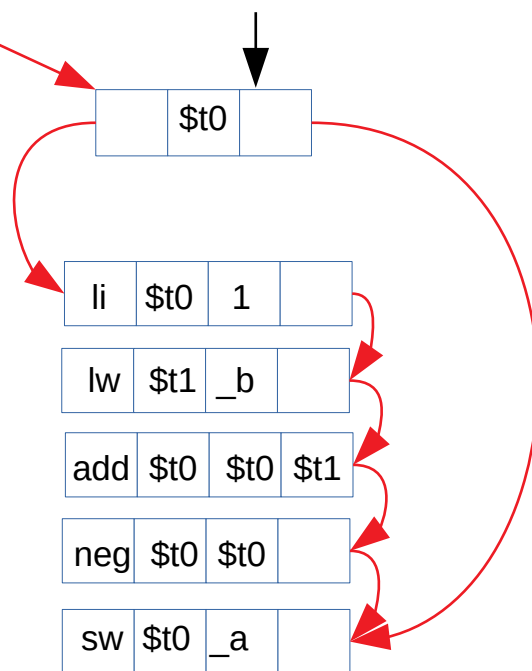
```
test1() {  
    var int a;  
    const int b = 3;  
  
    a = -(1 + b);  
  
}
```

registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

statement \rightarrow id = expression ; { ?? }

$\$ \$$ $\$1 \rightarrow a$ $\$3$



statement
id = expression ;

Generación de código de asignaciones (5)

```
test1() {  
    var int a;  
    const int b = 3;  
    a = -(1 + b);  
}
```

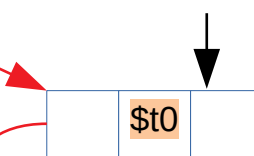
registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

statement \rightarrow id = expression ; { ?? }

\$\$ \$1 \rightarrow a \$3

¡No olvidéis
liberar el
registro de la
expresión!



li \$t0 1

lw \$t1 _b

add \$t0 \$t0 \$t1

neg \$t0 \$t0

sw \$t0 _a

statement
id = expression ;

Generación de código de print (1)

- Reglas de producción implicadas:

statement → PRINT (print_list) ;

print_list → print_item

 | print_list , print_item

print_item → expression

 | STRING

Generación de código de print (2)

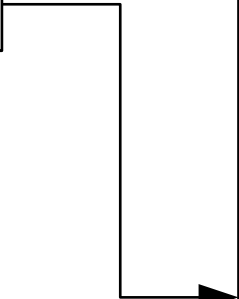
- Código MIPS para salida por consola de strings:

miniC

```
print ("Respuesta:");
```

MIPS

```
.data
...
$str1: .asciiz "Respuesta:"
...
.text
...
li $v0, 4
la $a0, $str1
syscall
```



Generación de código de print (3)

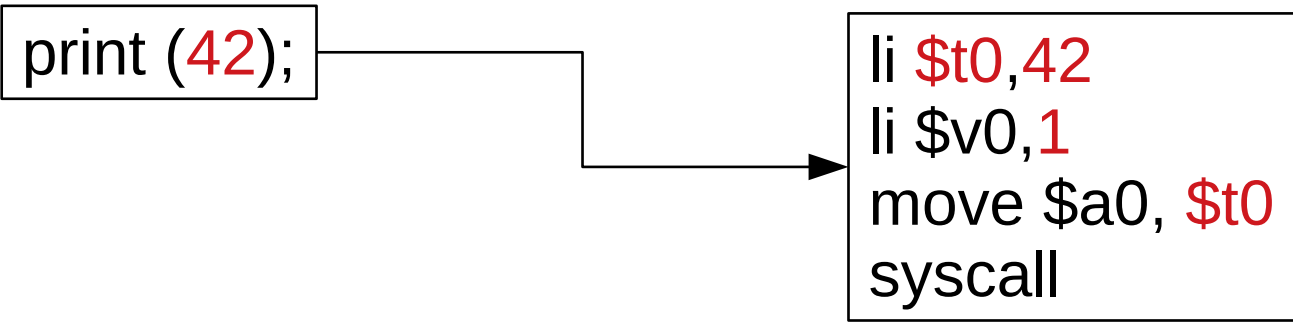
- Código MIPS para salida por consola de enteros:

miniC

```
print (42);
```

MIPS

```
li $t0,42  
li $v0,1  
move $a0, $t0  
syscall
```

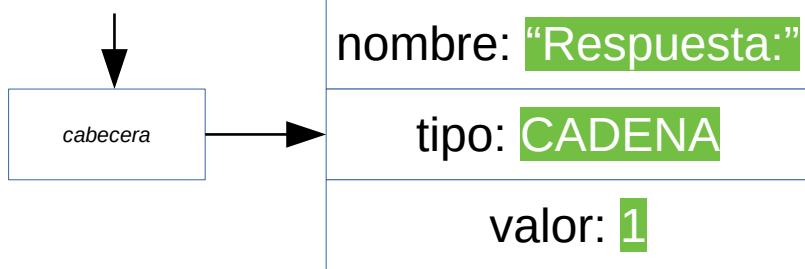


Generación de código de print (4)

```
test2() {  
    print ("Respuesta:",42);  
}
```

print_item → **STRING** { ?? }
 \$\$ \$1 → "Respuesta:"

símbolos

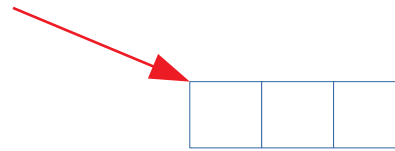


print_item
↑
STRING

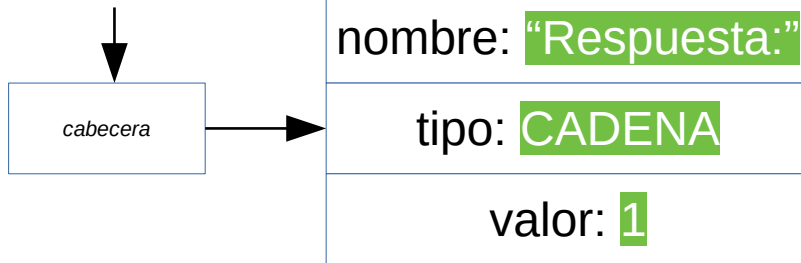
Generación de código de print (5)

```
test2() {  
    print ("Respuesta:",42);  
}
```

print_item → **STRING** { ?? }
\$\$ \$1 → "Respuesta:"



símbolos

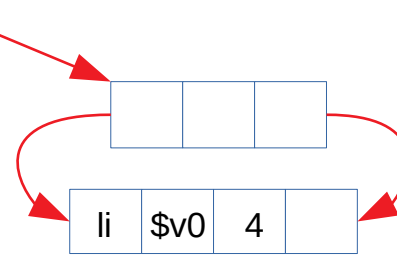


print_item
↑
STRING

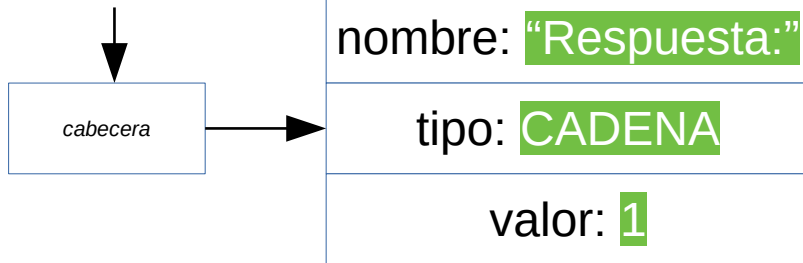
Generación de código de print (6)

```
test2() {  
    print ("Respuesta:",42);  
}
```

print_item → **STRING** { ?? }
\$\$ \$1 → "Respuesta:"



símbolos

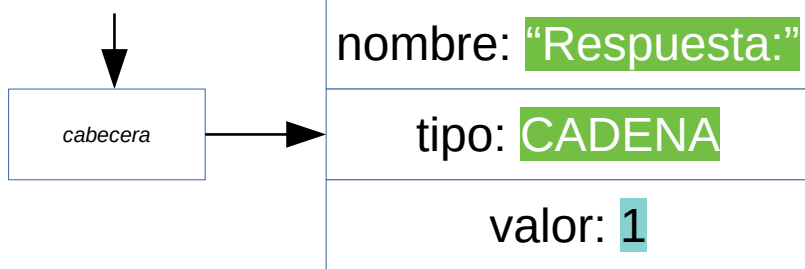


print_item
↑
STRING

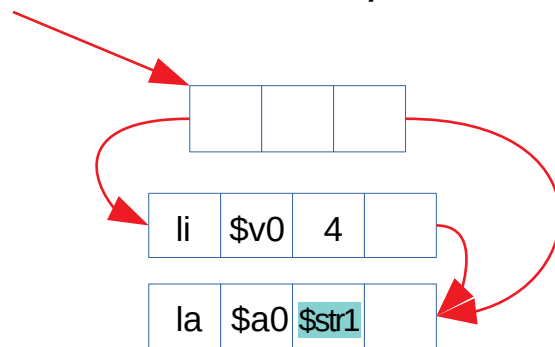
Generación de código de print (7)

```
test2() {  
    print ("Respuesta:",42);  
}
```

símbolos



print_item → **STRING** { ?? }
\$\$ \$1 → "Respuesta:"



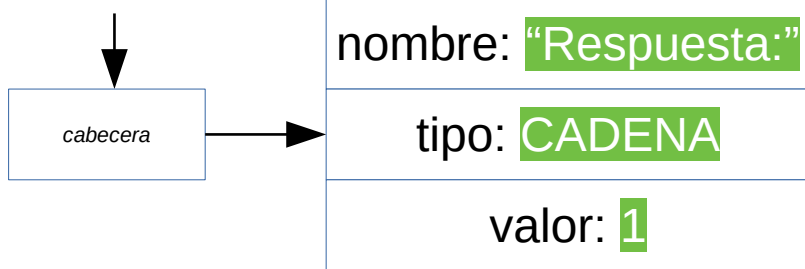
print_item

STRING

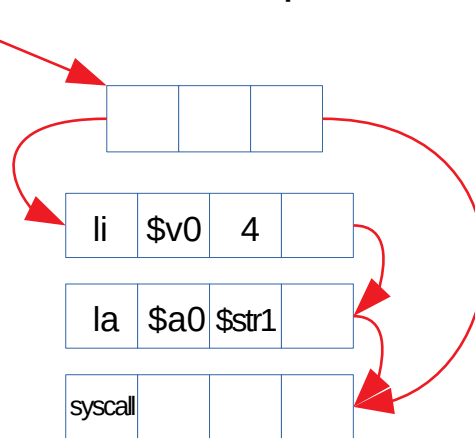
Generación de código de print (8)

```
test2() {  
    print ("Respuesta:",42);  
}
```

símbolos



print_item → **STRING** { ?? }
\$\$ \$1 → "Respuesta:"



print_item
↑
STRING

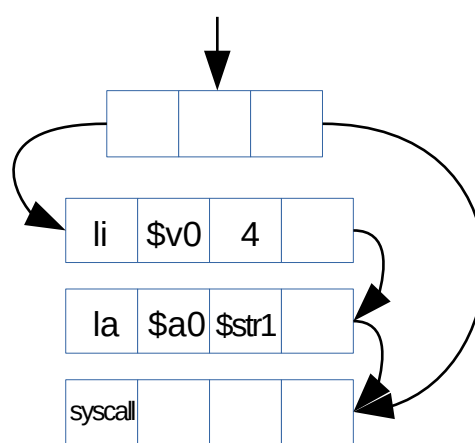
Generación de código de print (9)

```
test2() {  
    print ("Respuesta:",42);  
}
```

print_list → **print_item** { ?? }

\$\$

\$1



print_list

print_item

STRING

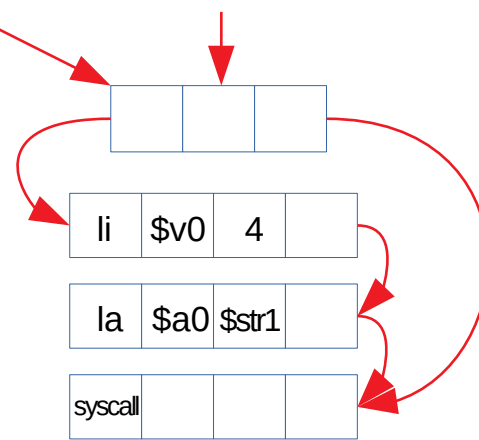
Generación de código de print (10)

```
test2() {  
    print ("Respuesta:",42);  
}
```

print_list → **print_item** { ?? }

\$\$

\$1



print_list
↑
print_item
↑
STRING

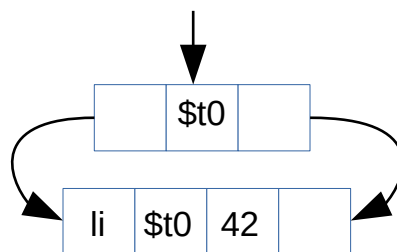
Generación de código de print (11)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

print_item → **expression** { ?? }

\$\$

\$1



registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

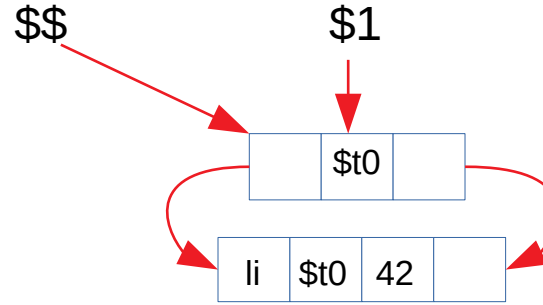
print_item

↑
expression

Generación de código de print (12)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

print_item → **expression** { ?? }



registros

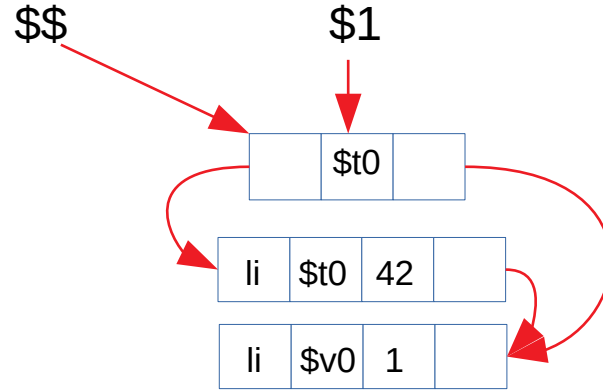
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

print_item
↑
expression

Generación de código de print (13)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

print_item → **expression** { ?? }



registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

print_item
↑
expression

Generación de código de print (14)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

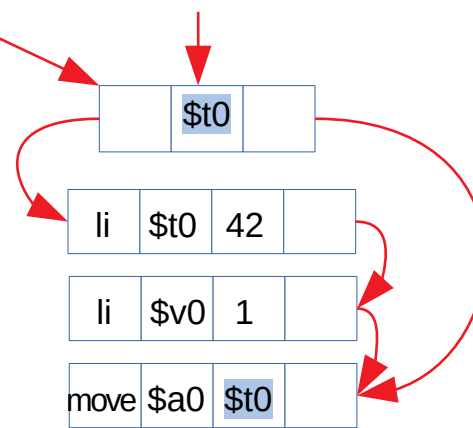
registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

print_item → **expression** { ?? }

\$\$

\$1



print_item
↑
expression

Generación de código de print (15)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

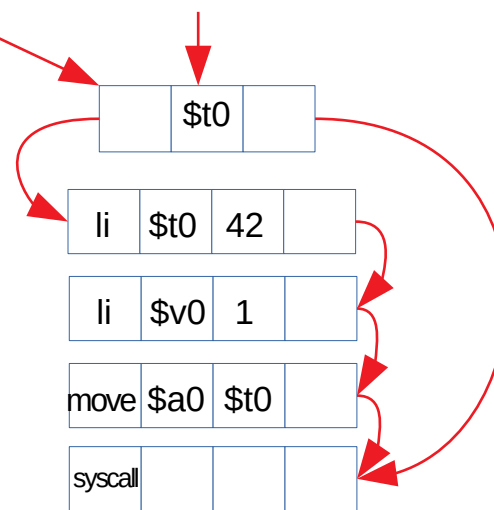
registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

print_item → **expression** { ?? }

\$\$

\$1



print_item
↑
expression

Generación de código de print (16)

```
test2() {  
    print ("Respuesta:", 42);  
}
```

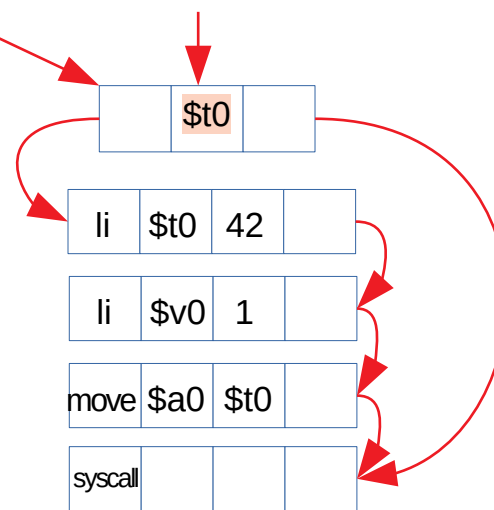
registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

print_item → **expression** { ?? }

\$\$

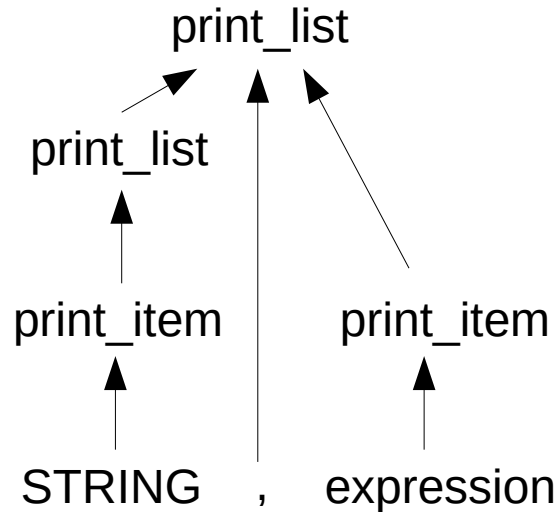
\$1



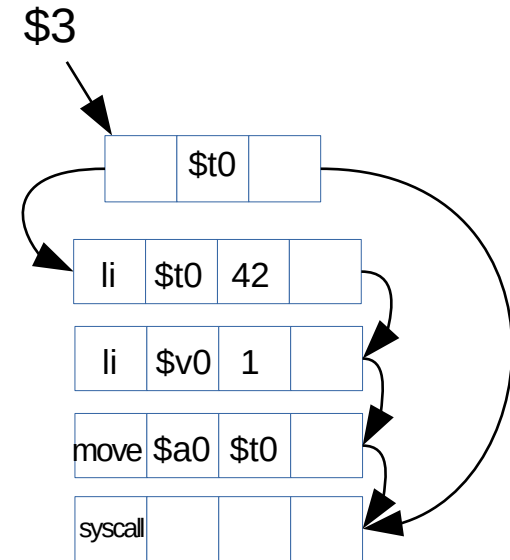
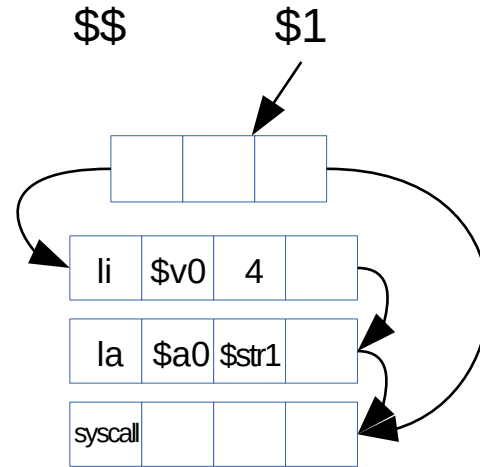
print_item
↑
expression

Generación de código de print (17)

```
test2() {  
    print ("Respuesta:",42);  
}
```

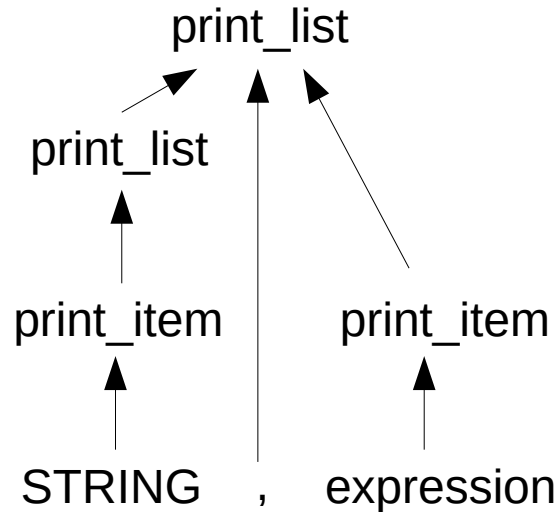


`print_list` → **`print_list`** , **`print_item`** { ?? }

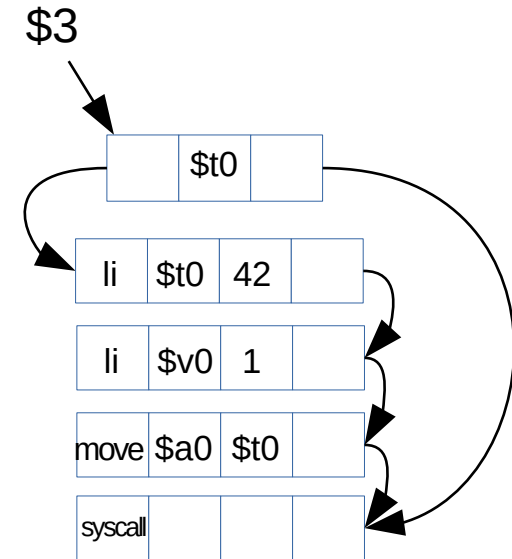
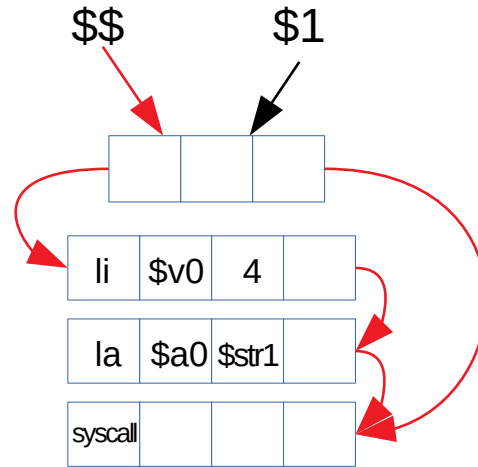


Generación de código de print (18)

```
test2() {  
    print ("Respuesta:",42);  
}
```

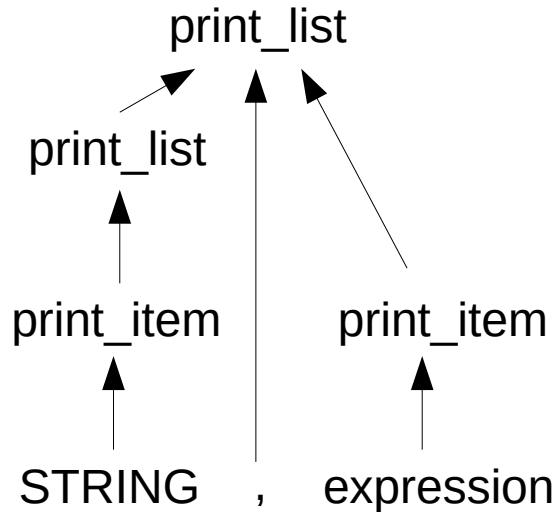


`print_list` → **`print_list`** , **`print_item`** { ?? }

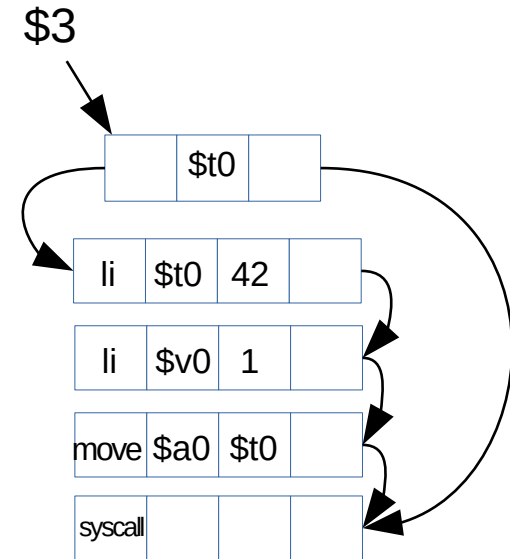
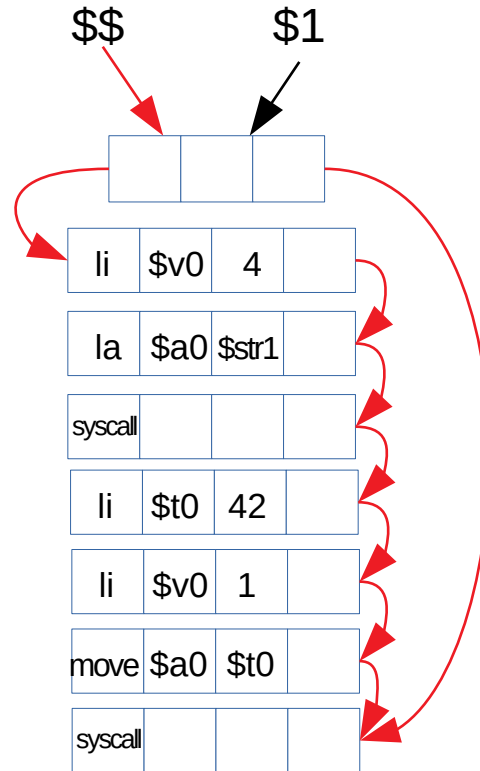


Generación de código de print (19)

```
test2() {  
    print ("Respuesta:",42);  
}
```

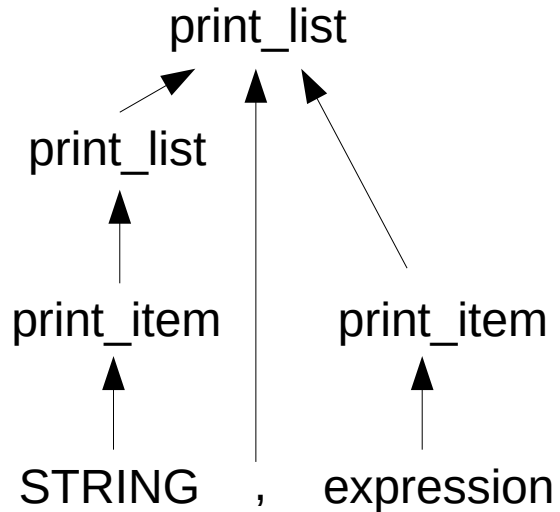


`print_list` → **`print_list`** , **`print_item`** { ?? }

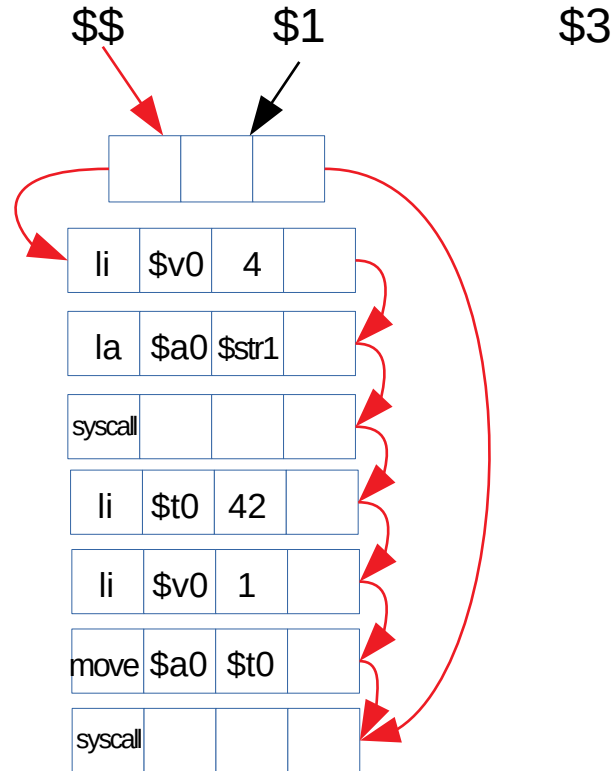


Generación de código de print (20)

```
test2() {  
    print ("Respuesta:",42);  
}
```

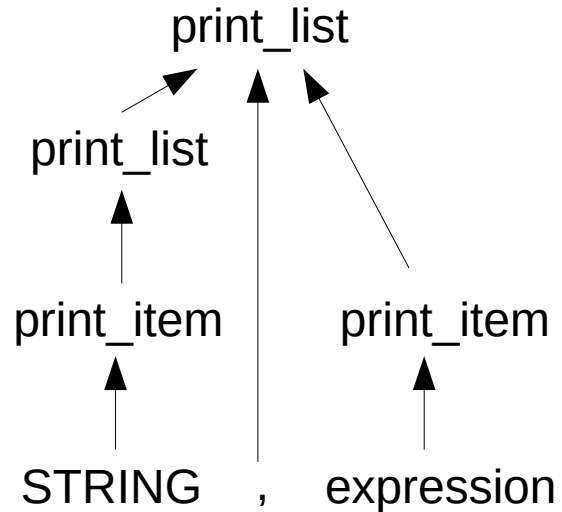


`print_list` → **`print_list`** , **`print_item`** { ?? }



Generación de código de print (21)

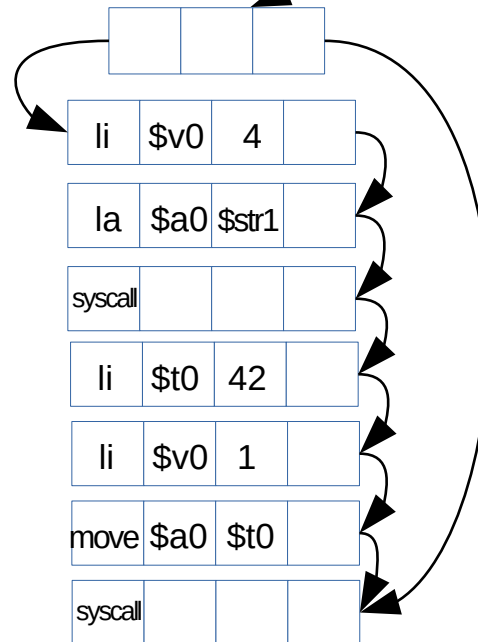
```
test2() {  
    print ("Respuesta:",42);  
}
```



statement \rightarrow PRINT (**print_list**) ; { ?? }

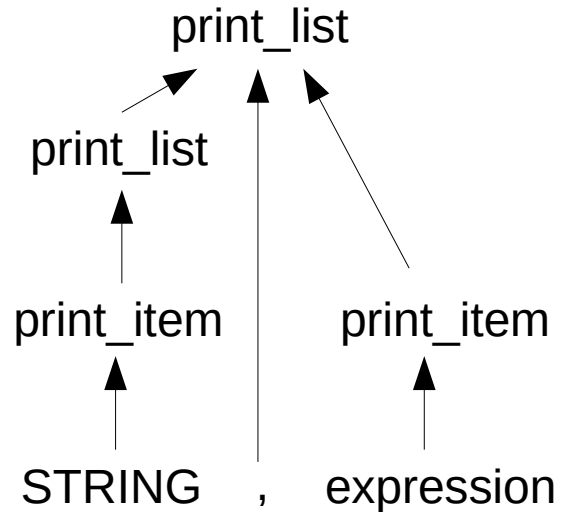
\$\$

\$3

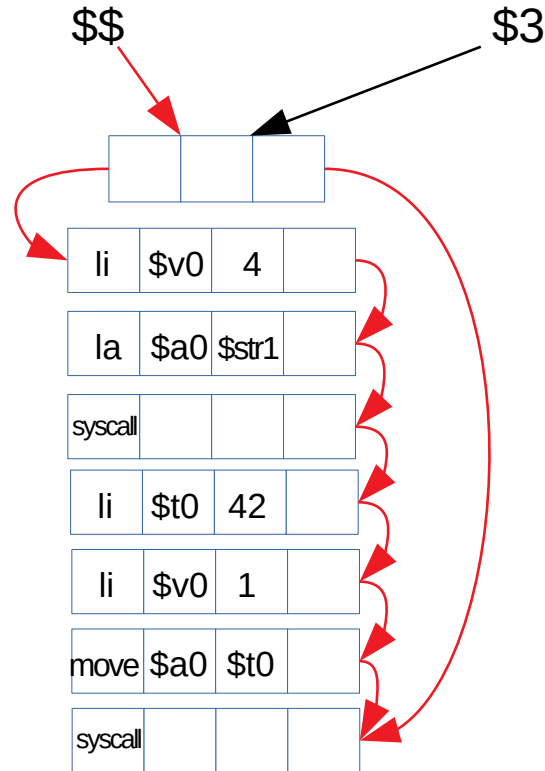


Generación de código de print (22)

```
test2() {  
    print ("Respuesta:",42);  
}
```



statement \rightarrow PRINT (**print_list**) ; { ?? }



Generación de código de read (1)

- Reglas de producción implicadas:

statement \rightarrow READ (read_list) ;

read_list \rightarrow id

| read_list , id

Generación de código de read (2)

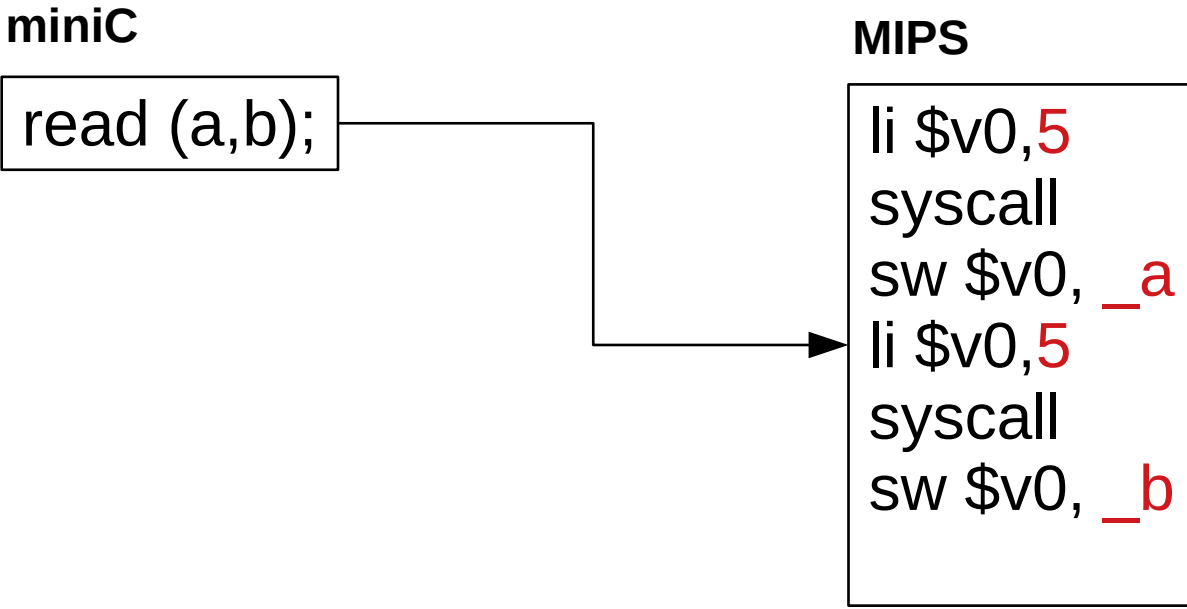
- Código MIPS para salida por consola de enteros:

miniC

```
read (a,b);
```

MIPS

```
li $v0,5  
syscall  
sw $v0, _a  
li $v0,5  
syscall  
sw $v0, _b
```

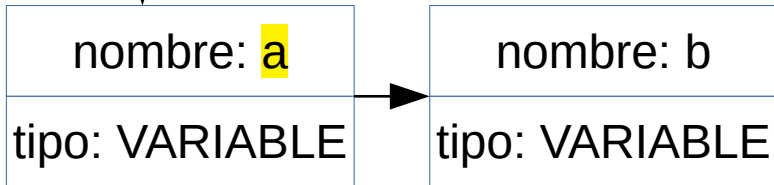
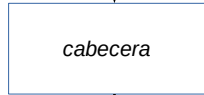


Generación de código de read (3)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

read_list \rightarrow id { ?? }
\$\$ \$1 \rightarrow "a"

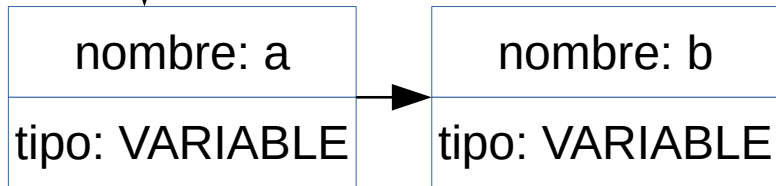
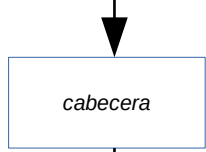
símbolos



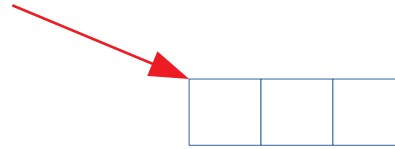
Generación de código de read (4)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos



read_list → **id** { ?? }
 \$\$ \$1 → "a"

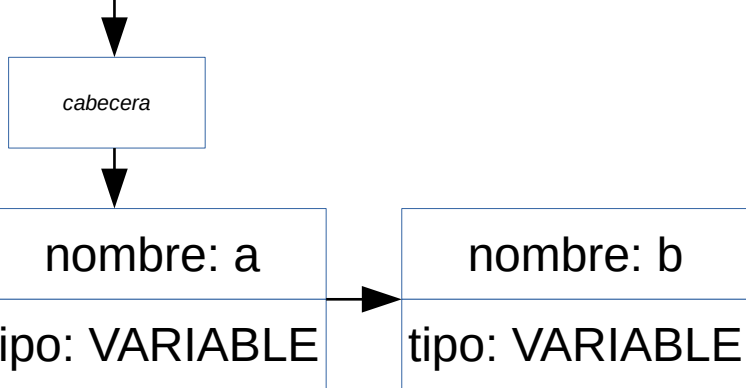


read_list
 ↑
 id

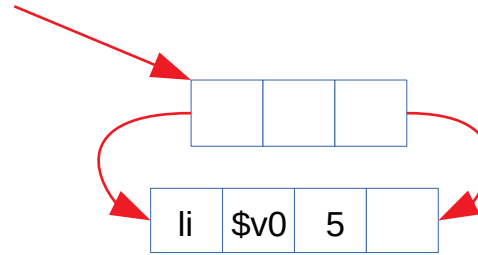
Generación de código de read (5)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos



read_list → **id** { ?? }
\$\$ \$1 → "a"



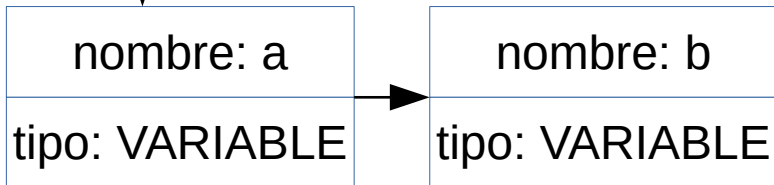
read_list
↑
id

Generación de código de read (6)

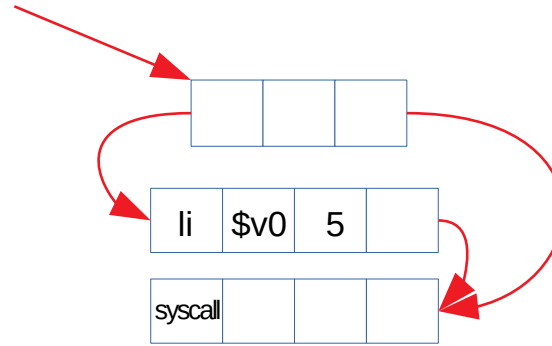
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



read_list → **id** { ?? }
\$\$ \$1 → "a"



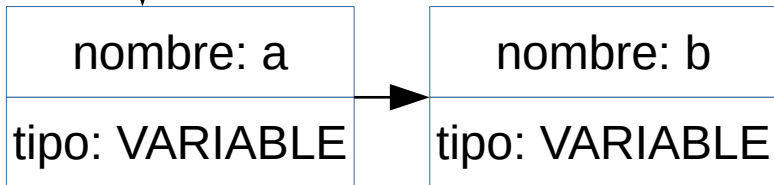
read_list
↑
id

Generación de código de read (7)

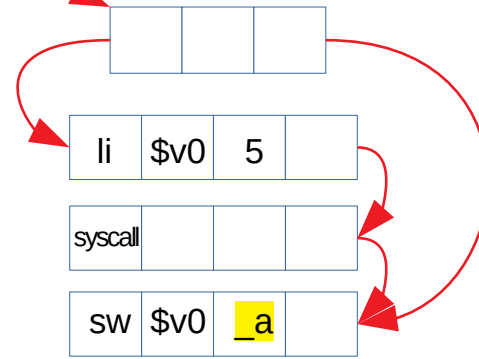
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



read_list → id { ?? }
\$\$ \$1 → "a"



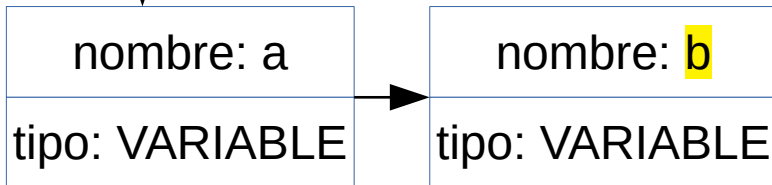
read_list
↑
id

Generación de código de read (8)

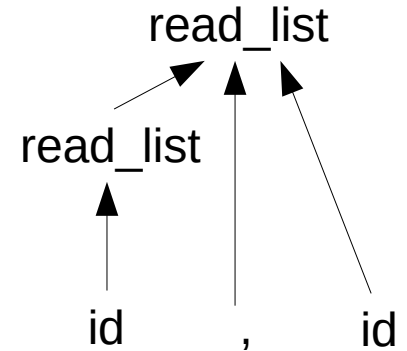
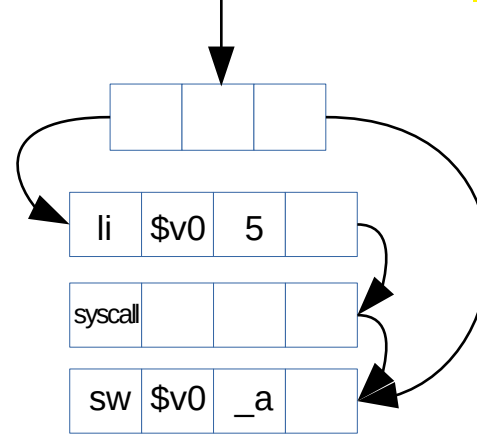
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



read_list → **read_list** , id { ?? }
 \$\$ \$1 \$3 → **"b"**



Generación de código de read (9)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera

nombre: a

tipo: VARIABLE

nombre: b

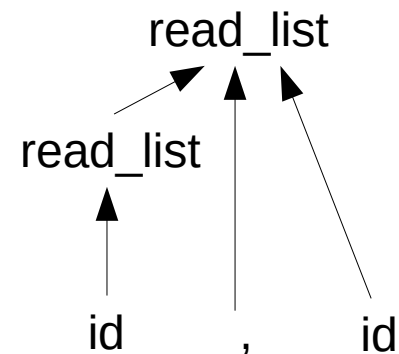
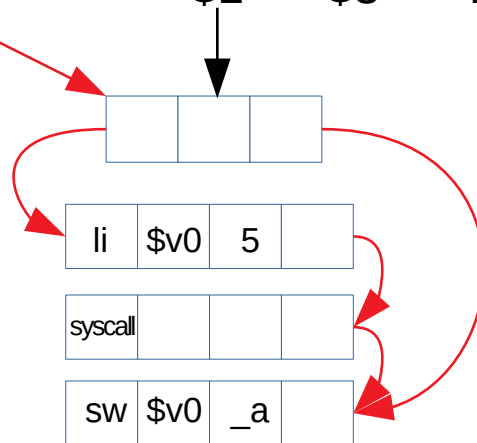
tipo: VARIABLE

read_list → **read_list** , id { ?? }

\$\$

\$1

\$3 → "b"

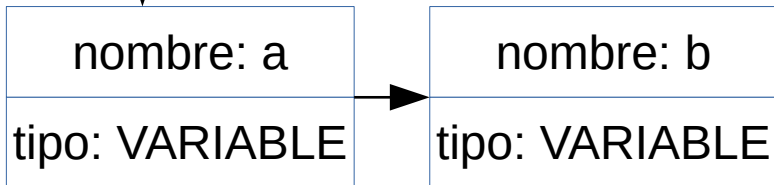


Generación de código de read (10)

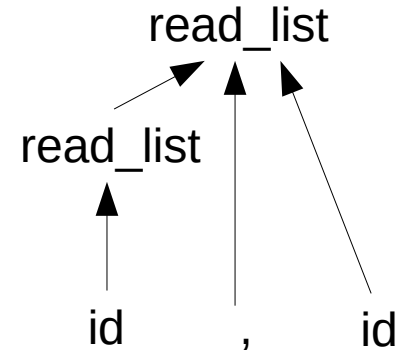
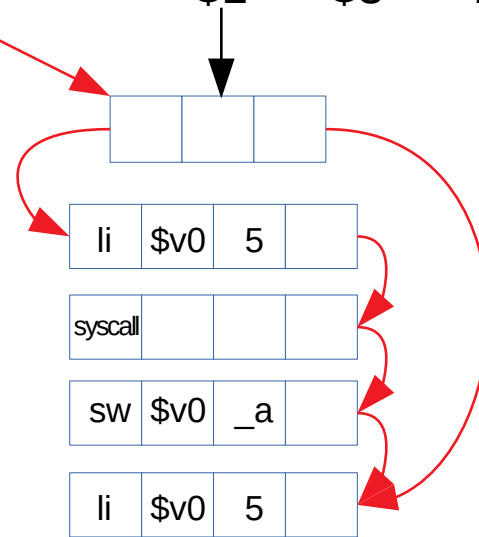
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



read_list → **read_list** , id { ?? }
\$\$ \$1 \$3 → "b"

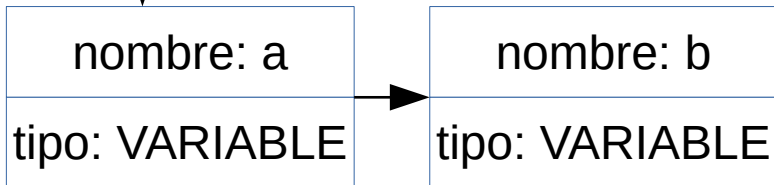


Generación de código de read (11)

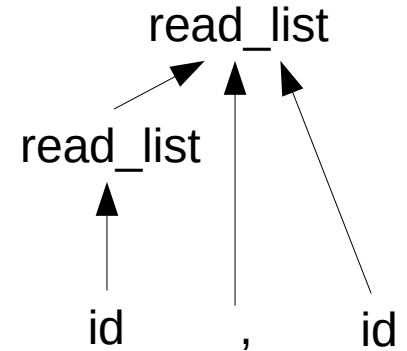
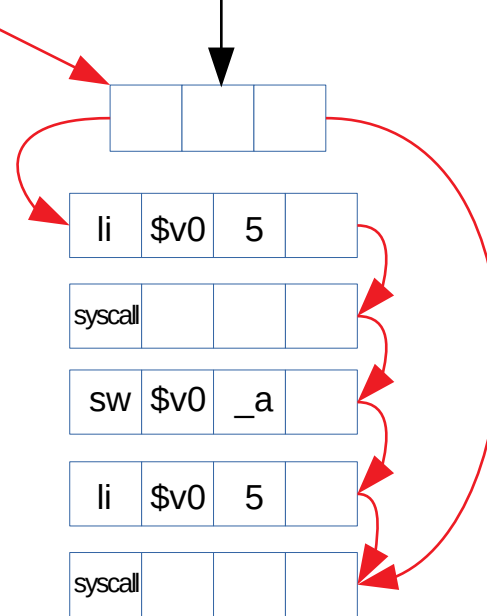
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



read_list → **read_list** , id { ?? }
\$\$ \$1 \$3 → "b"

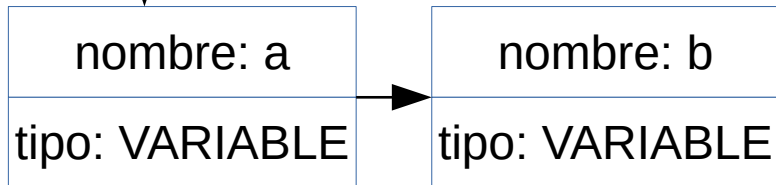


Generación de código de read (12)

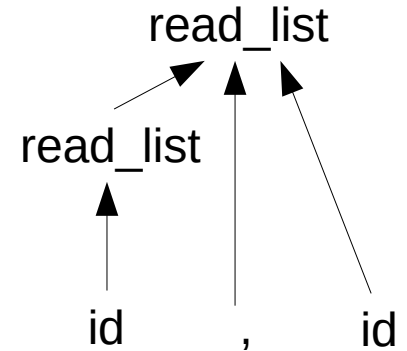
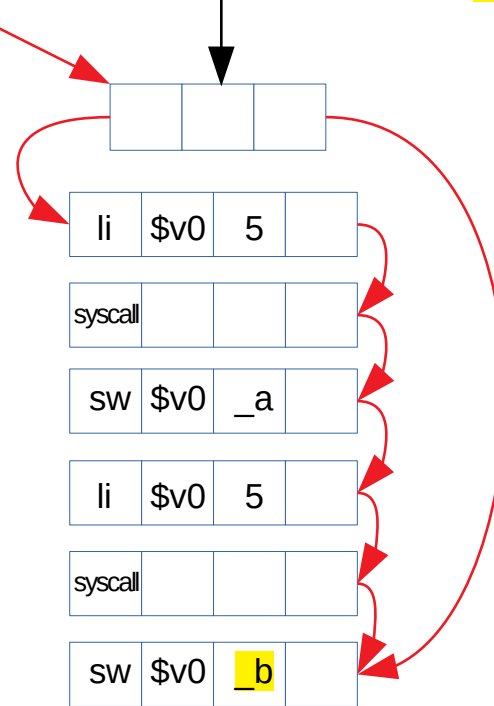
```
test3() {  
    var int a,b;  
  
    read ( a,b );  
}
```

símbolos

cabecera



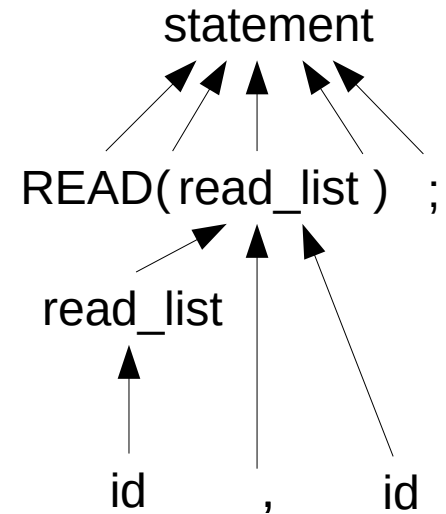
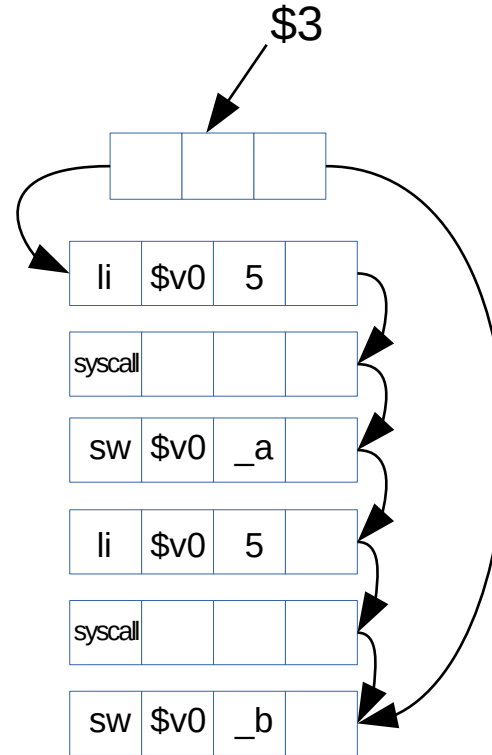
read_list → read_list , id { ?? }
\$\$ \$1 \$3 → "b"



Generación de código de read (13)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
  
}
```

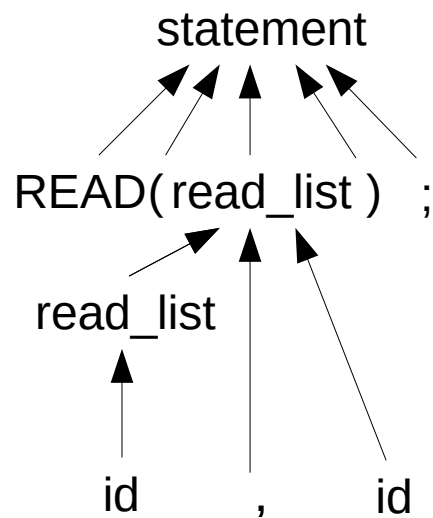
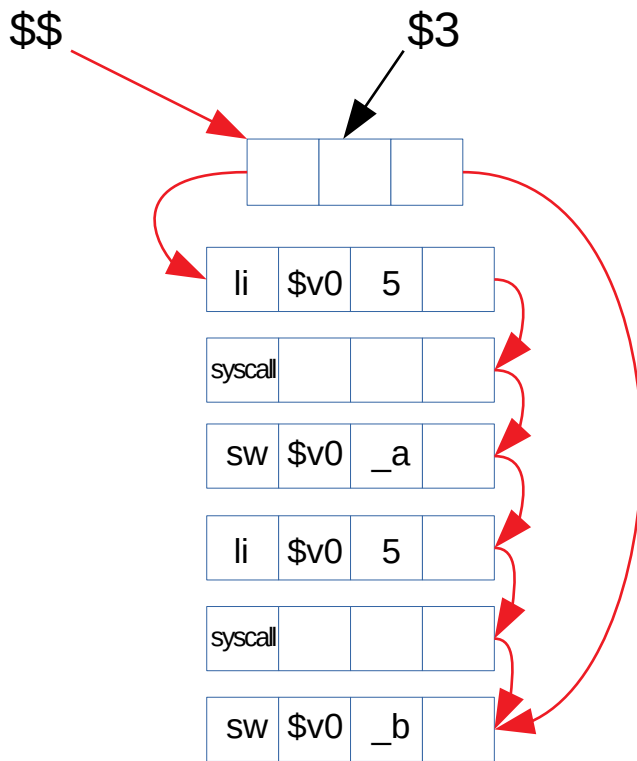
statement \rightarrow read (**read_list**); { ?? }
 \$\$ \$3



Generación de código de read (14)

```
test3() {  
    var int a,b;  
  
    read ( a,b );  
  
}
```

statement \rightarrow read (read_list) ; { ?? }



Condiciones

- En miniC, las condiciones son expresiones
- miniC no tienen operadores de comparación
- `if (expresion)` equivale a `if (expresion != 0)`
- ¡No olvidéis liberar el registro de la expresión!

Generación de código de if (1)

- Regla de producción implicada:

statement \rightarrow if (**expression**) **statement**

Generación de código de if (2)

- Código MIPS para if:

miniC

```
if (a) print ("a") ;
```

expression

statement

MIPS

```
.data
```

```
...
```

```
$str1: .asciiz "a"
```

```
...
```

```
.text
```

```
...
```

```
lw $t0, _a
```

```
beqz $t0, $l1
```

```
li $v0, 4
```

```
la $a0, $str1
```

```
syscall
```

```
$l1:
```

expression

statement

Generación de código de if-else (1)

- Regla de producción implicada:

statement \rightarrow if (**expression**) **statement** else **statement**

Generación de código de if-else (2)

- Código MIPS para if-else:

miniC

```
if (a) print ("a"); else a=1;
```

expression

statement

statement

MIPS

```
.data
...
$str1: .asciiz "a"
...
.text
...
    lw $t0, _a           } expression
    beqz $t0, $l1         }
    li $v0, 4             } statement
    la $a0, $str1
    syscall
    b $l2
    $l1:                  }
    li $t1, 1             } statement
    sw $t1, _a
    $l2:
```


Generación de código de while (1)

- Regla de producción implicada:

statement \rightarrow while (**expression**) **statement**

Generación de código de while (2)

- Código MIPS para while:

miniC

```
while (a)  a = a-1;
```

expression

statement

MIPS

```
.text
```

```
...
```

```
$l1:
```

```
lw $t0, _a
```

} expression

```
beqz $t0, $l2
```

```
lw $t1, _a
```

```
li $t2, 1
```

```
sub $t1, $t1, $t2
```

```
sw $t1, _a
```

} statement

```
b $l1
```

```
$l2:
```

Generación de código de statement_list (1)

- Reglas de producción implicadas:

statement $\rightarrow \{ \text{statement_list} \}$
statement_list $\rightarrow \text{statement_list statement}$
 | λ
 ↑
 %empty

Generación de código de statement_list (2)

- Ideas para la traducción:

```
while (a)
{
    print ("a=",a,"\n");
    a = a-1;
}
```

statement → { statement_list }
statement_list → statement_list statement

| λ

```
{ /* Pasar a $$ el código de $2 */      }
{ /* Pasar a $$ el código de $1 y
  concatenar en $$ el código de $2
  liberar código de $2 */
}
{ /* Poner en $$ lista código vacía */  }
```

Nos acercamos al final...

- En la regla de program se puede imprimir el código del statement_list:

```
program → id ( ) { declarations statement_list }  
  
{ /* Comprobar si todo ha ido bien, y en ese caso: */  
  imprimirLS(ts); // Volcar .data del programa  
  imprimirLC($6); // Volcar .text del programa  
}
```

ATENCIÓN: falta generar el código de declarations, con la inicialización de identificadores

¿Las declaraciones generan código?

- Sí, en la asignación de valores iniciales...

```
test4() {
```

```
    var int a;
```

```
    const int b = 3;
```

Generación de código de asignación

li \$t0, 3
sw \$t0, _b

```
    print ("b",b);
```

```
}
```

registros

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Reglas de producción usadas en la generación de código de declaraciones

declarations → declarations **var** tipo var_list ;
 | declarations **const** tipo const_list ;
 | λ

const_list → id = expression
 | const_list , id = expression

```
%type <codigo> ... declarations const_list
```

Ideas para const_list

```
const_list → id = expression {  
    /* 1. Verificación semántica de $1  
       2. $$ = código de asignación  
       3. Liberar registro de $3 */  
}  
  
| const_list , id = expression {  
    /* 1. Verificación semántica de $3  
       2. $$ = código de asignación  
       3. Liberar registro de $5 */  
}
```


Ideas para declarations

```
declarations → declarations var tipo var_list ; {  
    /* 1. Asignar código de $1 a $$ */  
}  
  
| declarations const tipo const_list ; {  
    /* 1. Asignar código de $1 a $$  
       2. Concatenar en $$ el código de $4  
       3. Liberar $4  
    */  
}  
  
| λ { /* Asignar a $$ lista de código vacía */ }
```

Finis coronat opus !

- En la regla de program:

```
program → id ( ) { declarations statement_list }  
    {  
        /* Comprobar si todo ha ido bien, y en ese caso: */  
        imprimirLS(ts);          // Volcar .data del programa  
        concatenaLC($5,$6);  
        imprimirLC($5); // Volcar .text del programa  
        /* Liberar memoria de tabla de símbolos y listas de código */  
    }
```

ATENCIÓN:

1. Si hay una acción en mitad de la regla, se le asigna un \$i y desplaza +1 el resto
2. ¡Hay que comprobar que el código ensamblador funciona en MARS o spim!

Final del programa MIPS

El método imprimirLC() debe concluir imprimiendo dos operaciones que realizan el exit del programa:

```
li $v0, 10  
syscall
```