Generación de código (II)
Traducción de sentencias

Dpto. de Ingeniería de la Información y las Comunicaciones

Índice de la explicación

- Traducción de `if`
- Traducción de `if-else`
- Traducción de `while`
- Traducción de `print`
- Traducción de `read`

Traducción de `if`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
x = 0;
...
}
```

Traducción de `if`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
x = 0;
...
}
```

Traducción de `if`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
x = 0;
...
}
```

$\longrightarrow$

Traducción de `if`

if (expr) stat

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
x = 0;
...
}
```

$\longrightarrow$

```
    lw $t0, _a
    li $t1, 2
    add $t2, $t0, $t1
    beqz $t2, $l1
    li $t0, 3
    sw $t0, _x
$l1: li $t0, 0
    sw $t0, _x
    ...
```

Traducción de `if-else`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
else x = 0;
...
}
```

Traducción de `if-else`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
else x = 0;
...
}
```

Traducción de `if-else`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
else x = 0;
...
}
```

$\longrightarrow$

Traducción de `if-else`

```
sent(){
var int x;
const int a = 1;
if (a+2) x = 3;
else x = 0;
...
}
```

$\longrightarrow$

if (expr) stat **else** stat

```
    lw $t0, _a
    li $t1, 2
    add $t2, $t0, $t1
    beqz $t2, $l1
    li $t0, 3
    sw $t0, _x
    b $l2
$l1: li $t0, 0
    sw $t0, _x
$l2: ...
```

Traducción de `while`

```
sent(){
var int x;
const int a = 1;
while (a+2) x = 3;
x = 0;
...
}
```

Traducción de `while`

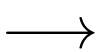
```
sent(){
var int x;
const int a = 1;
while (a+2) x = 3;
x = 0;
...
}
```

Traducción de `while`

```
sent(){
var int x;
const int a = 1;
while (a+2) x = 3;          ⟶
x = 0;
...
}
```

Traducción de `while`

```
sent(){
var int x;
const int a = 1;
while (a+2) x = 3;
x = 0;
...
}
```



while (expr) stat

$l1:	lw $t0, _a
	li $t1, 2
	add $t2, $t0, $t1
	beqz $t2, $l2
	li $t0, 3
	sw $t0, _x
	b $l1
$l2:	li $t0, 0
	sw $t0, _x
	...

Traducción de `print`

```
sent(){
var int x;
const int a = 1;
print (‘‘tres = ’’, a+2);
x = 0;
...
}
```

Traducción de `print`

```
sent(){
var int x;
const int a = 1;
print (‘‘tres = ’’, a+2);
x = 0;
...
}
```

Traducción de `print`

```
sent(){
var int x;
const int a = 1;
print (‘‘tres = ’’, a+2);
x = 0;
...
}
```

$\longrightarrow$

Traducción de `print`

```
sent(){
var int x;
const int a = 1;
print (‘‘tres = ’’, a+2);
x = 0;
...
}
```

$\longrightarrow$

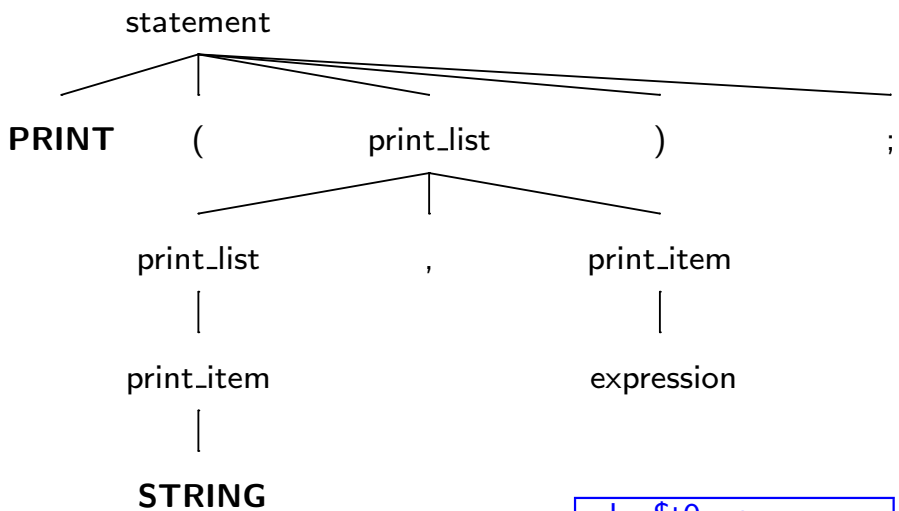
print...

$str1:

```
.asciiz “tres = ”
...
...
la $a0, $str1
li $v0, 4
syscall
lw $t0, _a
li $t1, 2
add $t2, $t0, $t1
move $a0, $t2
li $v0, 1
syscall
...
```

Uso y actualización de la `lista de código` en el fichero .y

Producción	**Qué debe realizar el código en C**
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1



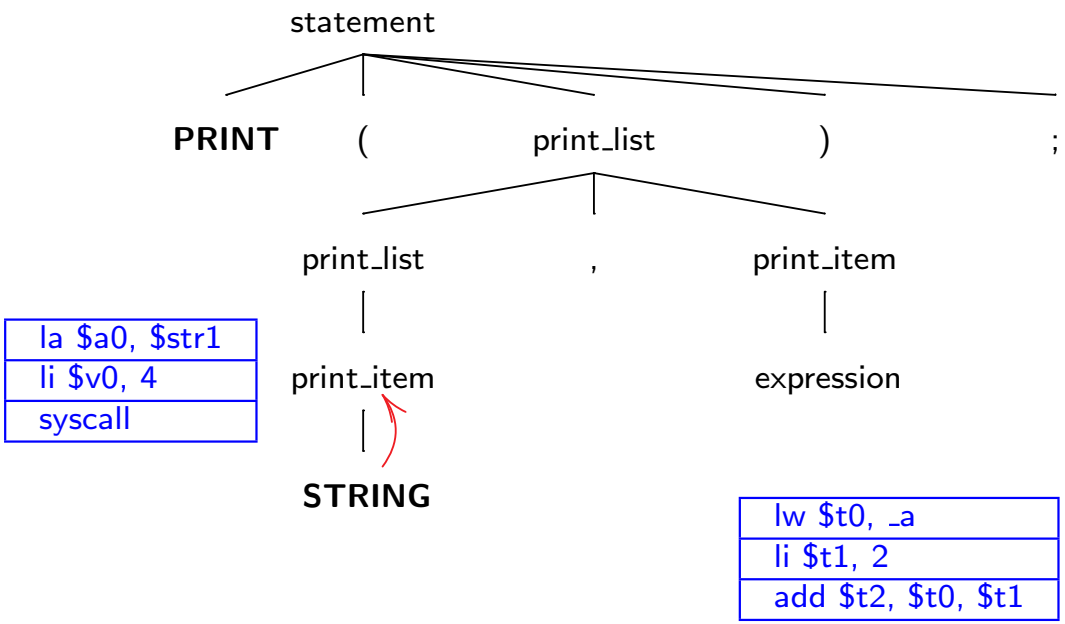

```
lw $t0, _a
li $t1, 2
add $t2, $t0, $t1
```

SENTENCIA A TRADUCIR:

print (“tres=” , a+2) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1

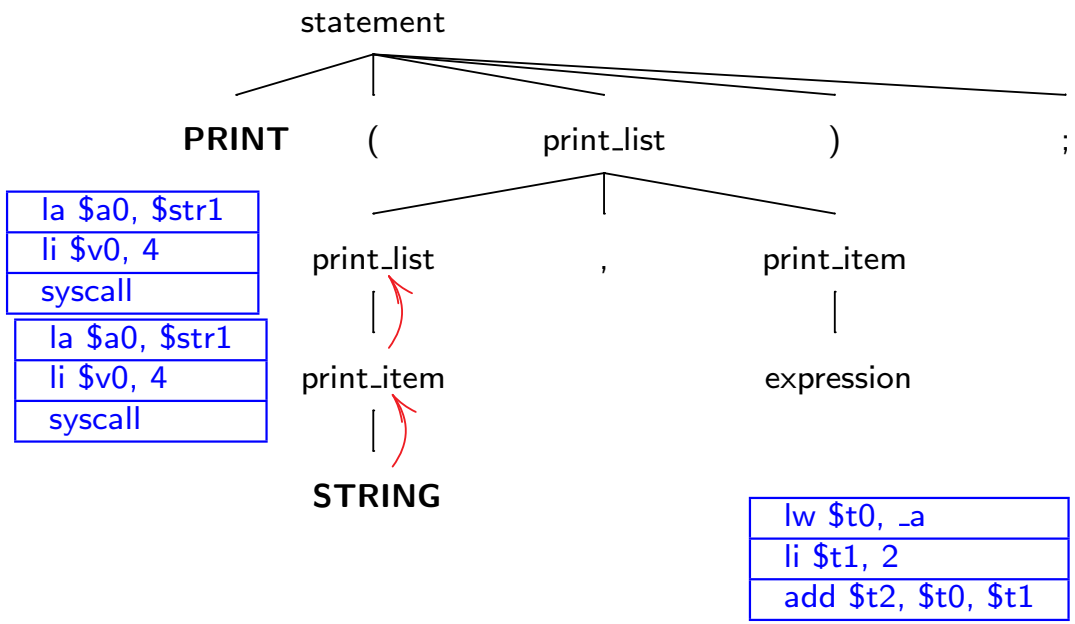


SENTENCIA A TRADUCIR:

print ("tres=" , a+2) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1

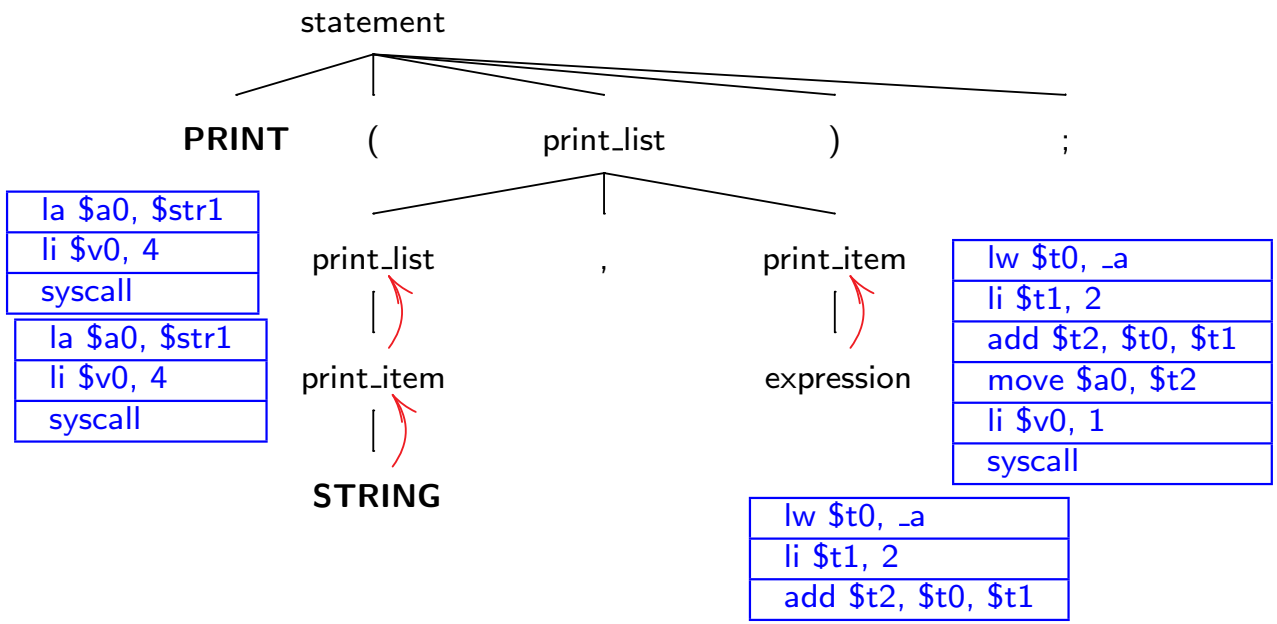


SENTENCIA A TRADUCIR:

print ("tres=" , a+2) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1

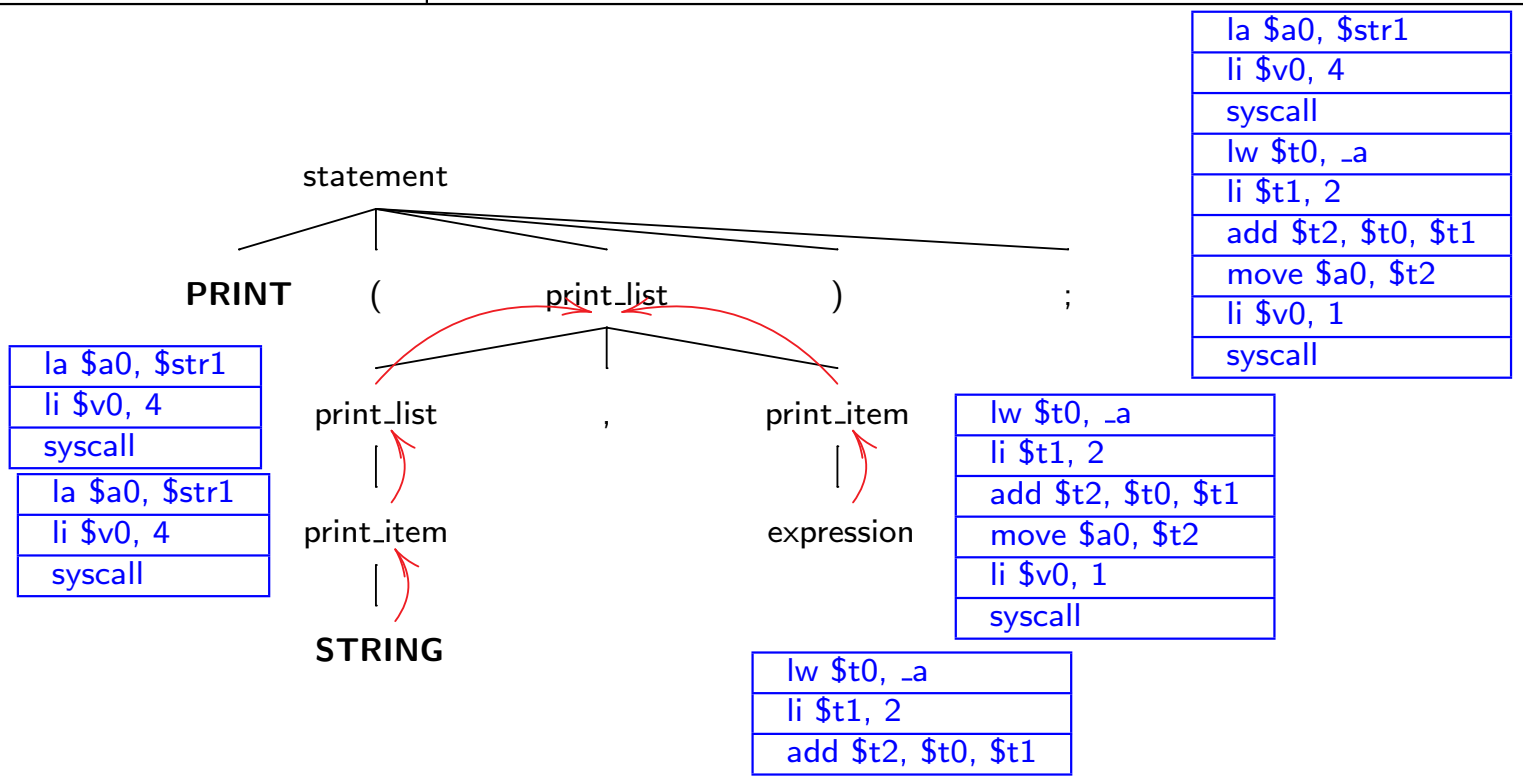


SENTENCIA A TRADUCIR:

print ("tres=" , a+2) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1

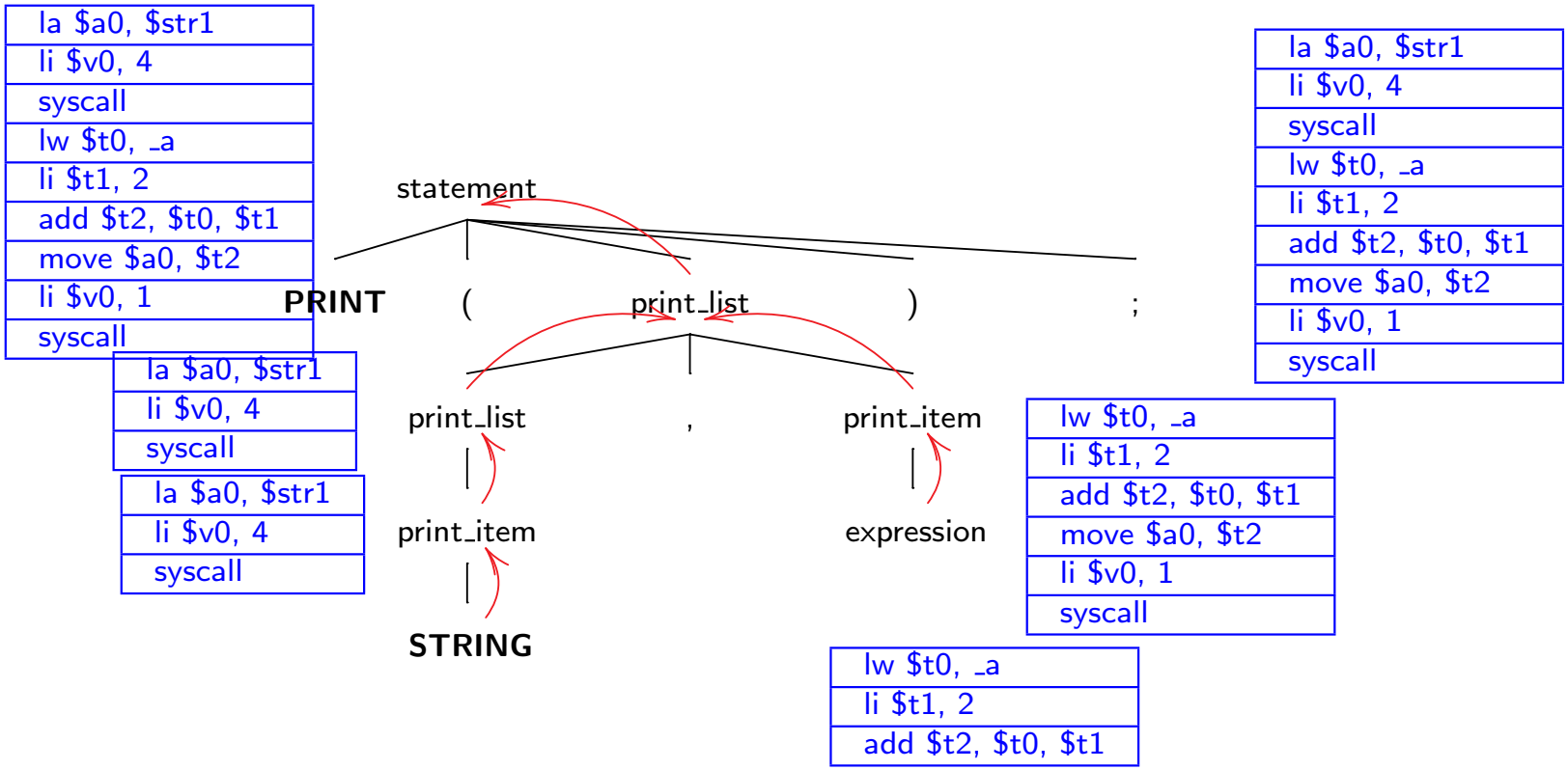


SENTENCIA A TRADUCIR:

print ("tres=" , a+2) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **PRINT** (print_list);	Asignar a $$ el contenido de $3
print_list→ print_item	Asignar a $$ el contenido de $1
| print_list , print_item	Asignar a $$ una ListaC con la concatenación de los códigos contenidos en $1 y $3
print_item → expression	Asignar a $$ el contenido de $1 seguido de las instrucciones de MIPS para imprimir el registro resultante de $1
| **STRING**	Asignar a $$ una ListaC con las instrucciones de MIPS para imprimir la cadena $1



SENTENCIA A TRADUCIR:

print (“tres=” , a+2) ;

Traducción de `read`

```
sent(){
var int x,y;
read (x, y);
x = 0;
...
}
```

Traducción de `read`

```
sent(){
var int x,y;
read (x, y);
x = 0;
...
}
```

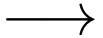
Traducción de `read`

```
sent(){
var int x,y;
read (x, y);
x = 0;
...
}
```

$\longrightarrow$

Traducción de `read`

```
sent(){
var int x,y;
read (x, y);
x = 0;
...
}
```

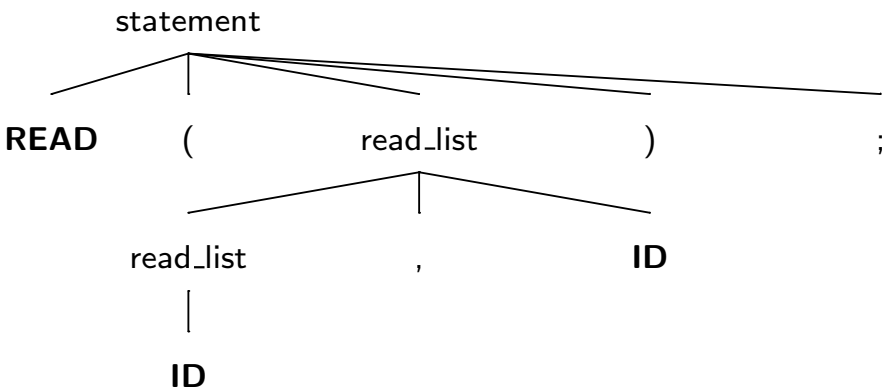


read...

li $v0, 5
syscall
sw $v0, _x
li $v0, 5
syscall
sw $v0, _y
...

Uso y actualización de la `lista de código` en el fichero .y

Producción	**Qué debe realizar el código en C**
statement → **READ** (read_list) ;	Asignar a $$ el contenido de $3
read_list→ **ID**	Asignar a $$ una ListaC con instrucciones MIPS para leer el valor del ID $1
| read_list , **ID**	Asignar a $$ una ListaC con la concatenación del código contenido en $1 y las instrucciones MIPS para leer el valor del ID $3

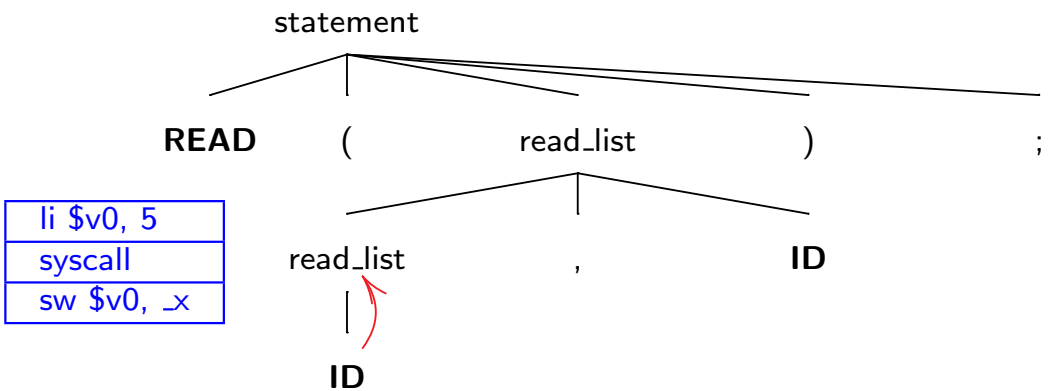


SENTENCIA A TRADUCIR:

read (x , y) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **READ** (read_list) ;	Asignar a $$ el contenido de $3
read_list→ **ID**	Asignar a $$ una ListaC con instrucciones MIPS para leer el valor del ID $1
| read_list , **ID**	Asignar a $$ una ListaC con la concatenación del código contenido en $1 y las instrucciones MIPS para leer el valor del ID $3

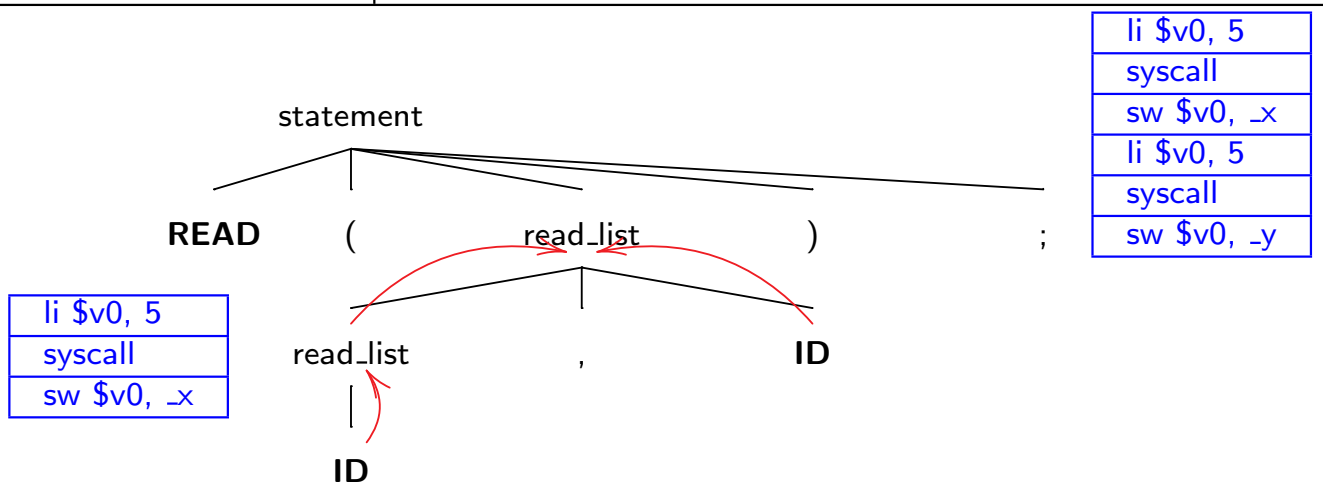


SENTENCIA A TRADUCIR:

read (x , y) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **READ** (read_list) ;	Asignar a $$ el contenido de $3
read_list→ **ID**	Asignar a $$ una ListaC con instrucciones MIPS para leer el valor del ID $1
| read_list , **ID**	Asignar a $$ una ListaC con la concatenación del código contenido en $1 y las instrucciones MIPS para leer el valor del ID $3

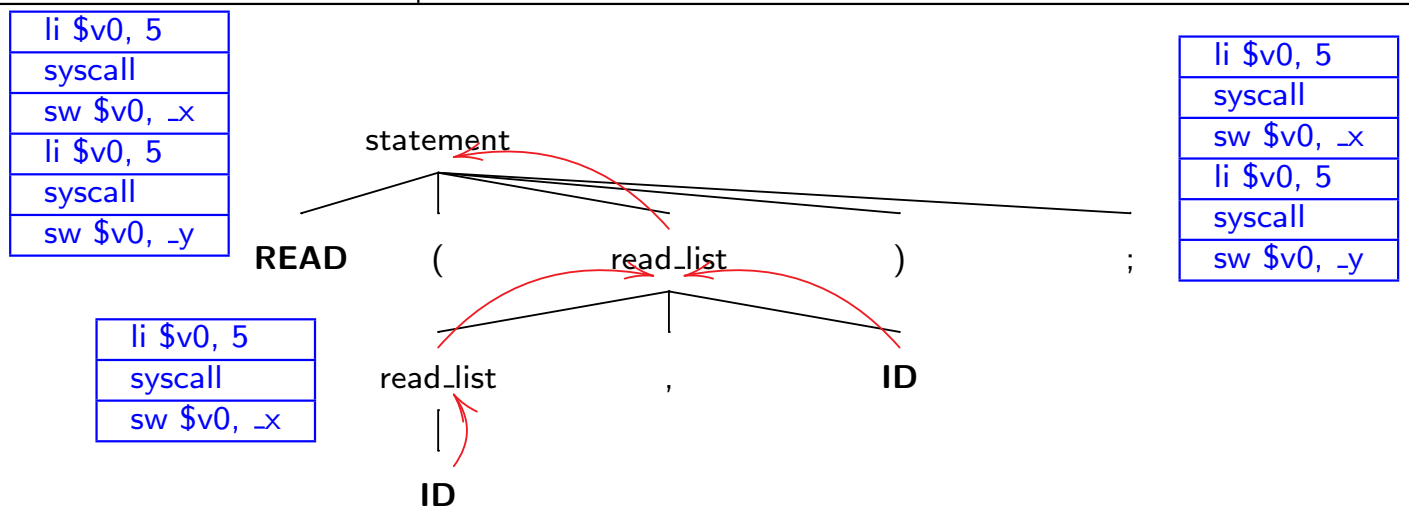


SENTENCIA A TRADUCIR:

read (x , y) ;

Uso y actualización de la `lista de código` en el fichero .y

Producción	Qué debe realizar el código en C
statement → **READ** (read_list) ;	Asignar a $$ el contenido de $3
read_list → **ID**	Asignar a $$ una ListaC con instrucciones MIPS para leer el valor del ID $1
| read_list , **ID**	Asignar a $$ una ListaC con la concatenación del código contenido en $1 y las instrucciones MIPS para leer el valor del ID $3



SENTENCIA A TRADUCIR:

read (x , y) ;

Uso y actualización de la `lista de código` en el fichero .y

Para lograr todo esto necesitaremos:

- Indicar a Bison que **los atributos de los no terminales que derivan sentencias serán también de tipo** `codigo`:

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

- Declarar e inicializar en la cabecera del fichero .y una variable global para ir generando las etiquetas asociadas a cada sentencia:

```
%{
...
int contadorEtiquetas=1;
...
%}
```

Para convertir el número de etiqueta `i` en la cadena ‘‘`$li`’’ puede sernos útil una función como la siguiente, que podemos implementar en la última sección del fichero .y:

```
char *obtenerEtiqueta() {
        char aux[32];
        sprintf(aux,"$l%d",contadorEtiquetas++);
        return strdup(aux);
}
```