Generación de código (I)
Traducción de expresiones

Dpto. de Ingeniería de la Información y las Comunicaciones

Índice de la explicación

- Objetivo
- Herramienta para alcanzar el objetivo: `lista de código`
- Uso y actualización de la `lista de código` en el fichero .y

Objetivo

```
expr() {
var int x;
const int a = 3;
x = a - 2 * 4;
}
```

Objetivo

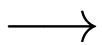
```
expr() {
var int x;
const int a = 3;
x = a - 2 * 4;
}
```

Objetivo

```
expr() {
var int x;
const int a = 3;        ⟶
x = a - 2 * 4;
}
```

Objetivo

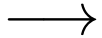
```
expr() {
var int x;
const int a = 3;
x = a - 2 * 4;
}
```



```
lw $t0, _a
li $t1, 2
li $t2, 4
mul $t3, $t1, $t2
sub $t1, $t0, $t3
sw $t1, _x
```

Objetivo

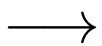
```
expr() {
var int x;
const int a = 3;
x = a - 2 * 4;
}
```



```
lw $t0, _a
li $t1, 2
li $t2, 4
mul $t3, $t1, $t2
sub $t1, $t0, $t3
sw $t1, _x
```

Objetivo

```
expr() {
var int x;
const int a = 3;
x = a - 2 * 4;
}
```

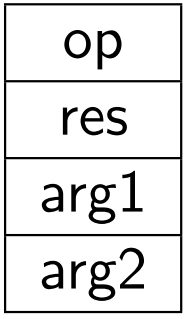


```
lw $t0, _a
li $t1, 2
li $t2, 4
mul $t3, $t1, $t2
sub $t1, $t0, $t3
sw $t1, _x
```

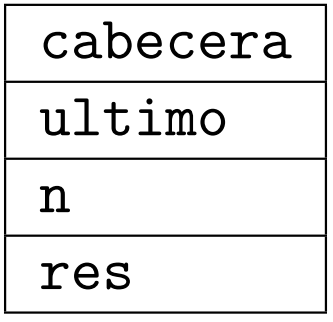
Herramienta para alcanzar el objetivo: `lista de código`

En "Prácticas/lista para código" podéis encontrar los ficheros `listaCodigo.h` y `listaCodigo.c`.

```
typedef struct {
char * op;
char * res;
char * arg1;
char * arg2;
} Operacion;
```



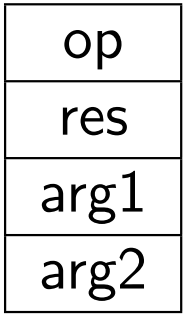

```
struct ListaCRep {
PosicionListaC cabecera;
PosicionListaC ultimo;
int n;
char *res;
};
```



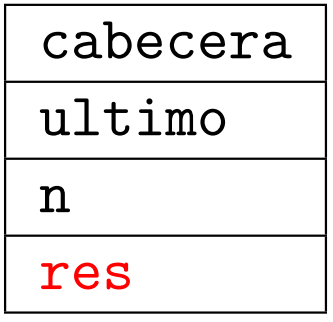
Herramienta para alcanzar el objetivo: `lista de código`

En "Prácticas/lista para código" podéis encontrar los ficheros `listaCodigo.h` y `listaCodigo.c`.

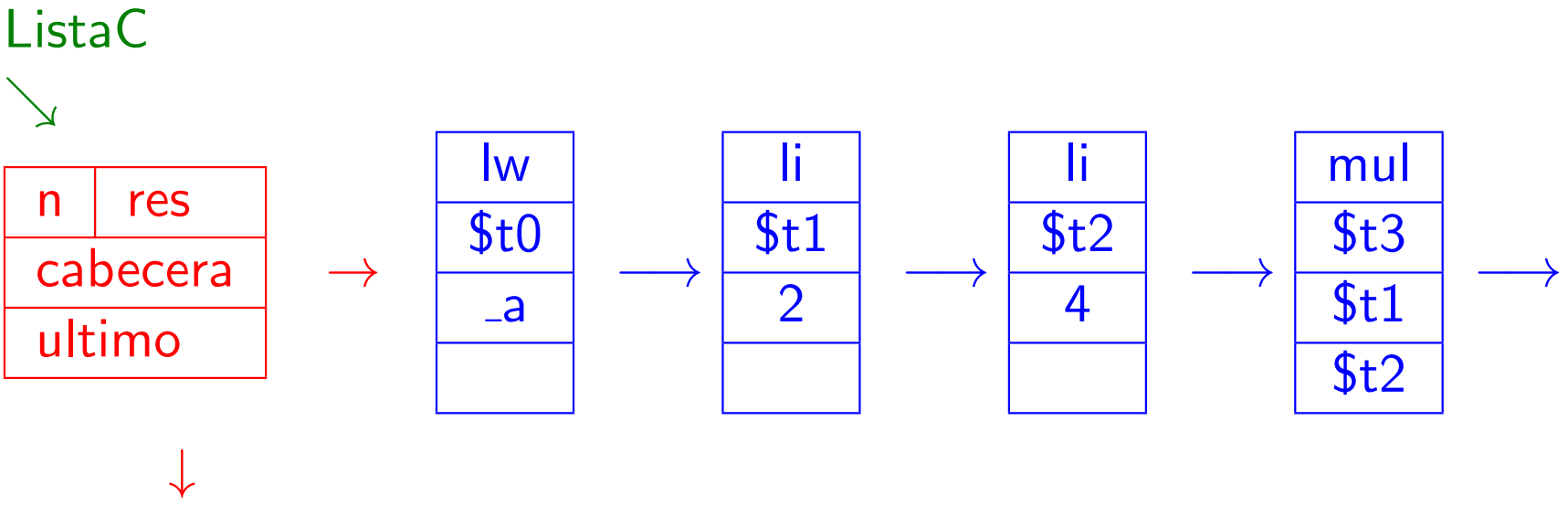
```
typedef struct {
char * op;
char * res;
char * arg1;
char * arg2;
} Operacion;
```




```
struct ListaCRep {
PosicionListaC cabecera;
PosicionListaC ultimo;
int n;
char *res;
};
```



Herramienta para alcanzar el objetivo: `lista de código`



Para almacenar el código:

```
lw $t0, _a
li $t1, 2
li $t2, 4
mul $t3, $t1, $t2
sub $t1, $t0, $t3
```

Herramienta para alcanzar el objetivo: `lista de código`

Y disponemos de las funciones siguientes para manipular la lista de código:

```
ListaC creaLC();
void liberaLC(ListaC codigo);
void insertaLC(ListaC codigo, PosicionListaC p, Operacion o);
Operacion recuperaLC(ListaC codigo, PosicionListaC p);
PosicionListaC buscaLC(ListaC codigo, PosicionListaC p, char *clave, Campo
campo);
void asignaLC(ListaC codigo, PosicionListaC p, Operacion o);
void concatenaLC(ListaC codigo1, ListaC codigo2);
int longitudLC(ListaC codigo);
PosicionListaC inicioLC(ListaC codigo);
PosicionListaC finalLC(ListaC codigo);
PosicionListaC siguienteLC(ListaC codigo, PosicionListaC p);
void guardaResLC(ListaC codigo, char *res);
char * recuperaResLC(ListaC codigo);
```

Herramienta para alcanzar el objetivo: `lista de código`

Y disponemos de las funciones siguientes para manipular la lista de código:

```
ListaC creaLC();
void liberaLC(ListaC codigo);
void insertaLC(ListaC codigo, PosicionListaC p, Operacion o);
Operacion recuperaLC(ListaC codigo, PosicionListaC p);
PosicionListaC buscaLC(ListaC codigo, PosicionListaC p, char *clave, Campo
campo);
void asignaLC(ListaC codigo, PosicionListaC p, Operacion o);
void concatenaLC(ListaC codigo1, ListaC codigo2);
int longitudLC(ListaC codigo);
PosicionListaC inicioLC(ListaC codigo);
PosicionListaC finalLC(ListaC codigo);
PosicionListaC siguienteLC(ListaC codigo, PosicionListaC p);
void guardaResLC(ListaC codigo, char *res);
char * recuperaResLC(ListaC codigo);
```

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

Debemos preparar nuestros ficheros para la traducción:

- Para poder usar las estructuras y funciones de `listaCodigo`:
  - → O bien añadimos en la cabecera del .y y en la cabecera del .l, entre %{ y %}:

```
#include ‘‘listaCodigo.h’’
```

  - → O bien añadimos sólo en la cabecera del .y:

```
%code requires{
#include ‘‘listaCodigo.h’’
}
```

- Además, para que los atributos de las expresiones sean de tipo `ListaC`, tenemos que añadir en la cabecera del .y:

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

```
%type <codigo> expression
```

- Finalmente, debemos adaptar el fichero `makefile` a la nueva situación, incluyendo el fichero `listaCodigo.c` en la compilación.

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

Debemos preparar nuestros ficheros para la traducción:

- Para poder usar las estructuras y funciones de `listaCodigo`:
  - → O bien añadimos en la cabecera del .y y en la cabecera del .l, entre %{ y %}:

```
#include ‘‘listaCodigo.h’’
```

  - → O bien añadimos sólo en la cabecera del .y:

```
%code requires{
#include ‘‘listaCodigo.h’’
}
```

- Además, para que los atributos de las expresiones sean de tipo `ListaC`, tenemos que añadir en la cabecera del .y:

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

```
%type <codigo> expression
```

- Finalmente, debemos adaptar el fichero `makefile` a la nueva situación, incluyendo el fichero `listaCodigo.c` en la compilación.

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

Debemos preparar nuestros ficheros para la traducción:

- Para poder usar las estructuras y funciones de `listaCodigo`:
  - → O bien añadimos en la cabecera del .y y en la cabecera del .l, entre %{ y %}:

```
#include ‘‘listaCodigo.h’’
```

  - → O bien añadimos sólo en la cabecera del .y:

```
%code requires{
#include ‘‘listaCodigo.h’’
}
```

- Además, para que los atributos de las expresiones sean de tipo `ListaC`, tenemos que añadir en la cabecera del .y:

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

```
%type <codigo> expression
```

- Finalmente, debemos adaptar el fichero `makefile` a la nueva situación, incluyendo el fichero `listaCodigo.c` en la compilación.

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

Debemos preparar nuestros ficheros para la traducción:

- Para poder usar las estructuras y funciones de `listaCodigo`:
  - → O bien añadimos en la cabecera del .y y en la cabecera del .l, entre %{ y %}:

```
#include ‘‘listaCodigo.h’’
```

  - → O bien añadimos sólo en la cabecera del .y:

```
%code requires{
#include ‘‘listaCodigo.h’’
}
```

- Además, para que los atributos de las expresiones sean de tipo `ListaC`, tenemos que añadir en la cabecera del .y:

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

```
%type <codigo> expression
```

- Finalmente, debemos adaptar el fichero `makefile` a la nueva situación, incluyendo el fichero `listaCodigo.c` en la compilación.

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

Debemos preparar nuestros ficheros para la traducción:

- Para poder usar las estructuras y funciones de `listaCodigo`:
  - → O bien añadimos en la cabecera del .y y en la cabecera del .l, entre %{ y %}:

```
#include ‘‘listaCodigo.h’’
```

  - → O bien añadimos sólo en la cabecera del .y:

```
%code requires{
#include ‘‘listaCodigo.h’’
}
```

- Además, para que los atributos de las expresiones sean de tipo `ListaC`, tenemos que añadir en la cabecera del .y:

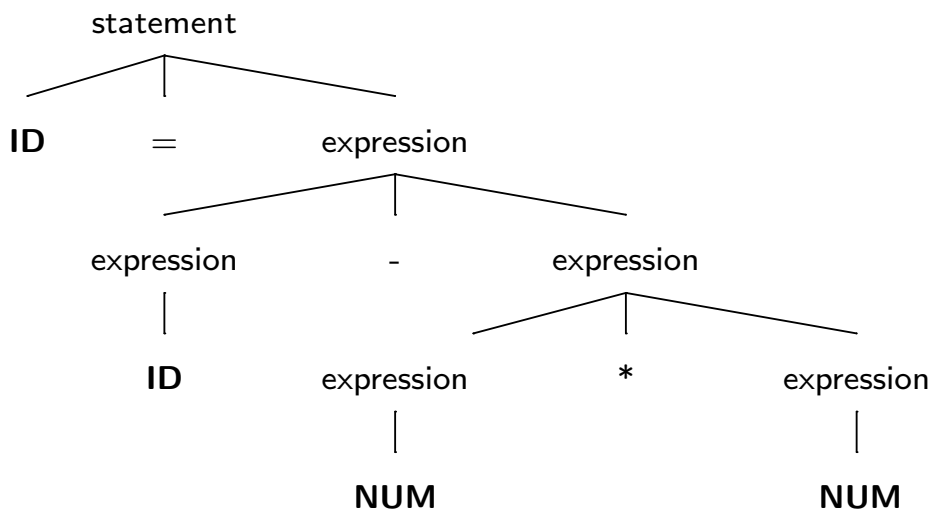
```
%union{
char *lexema;
ListaC codigo;
}
```

```
%type <codigo> expression
```

- Finalmente, debemos adaptar el fichero `makefile` a la nueva situación, incluyendo el fichero `listaCodigo.c` en la compilación.

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

Producción	**Qué debe realizar el código en C**
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
| $-E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
| (E_1)	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

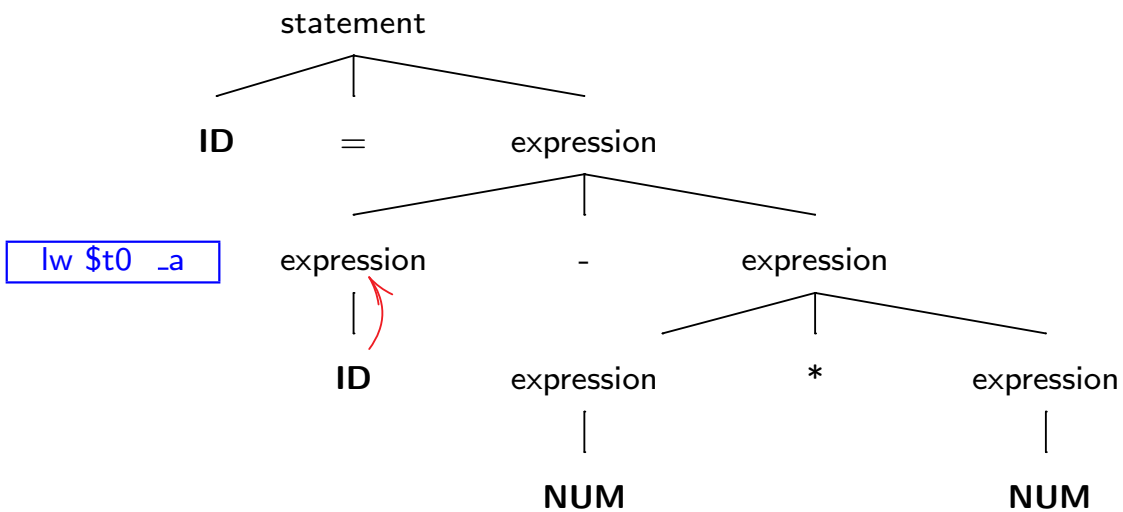
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
$\mid -E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
$\mid (E_1)$	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

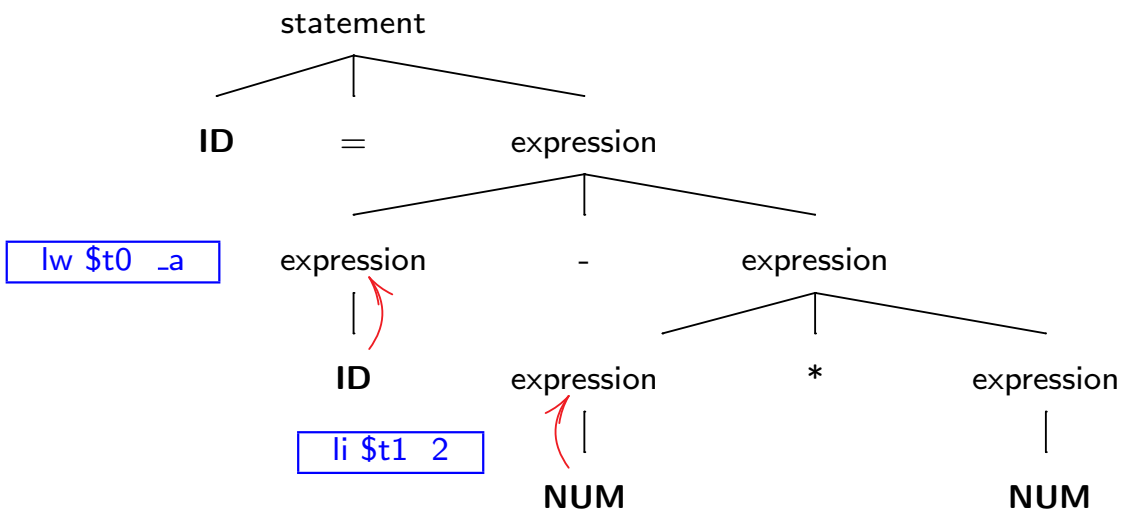
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
$\mid -E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
$\mid (E_1)$	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

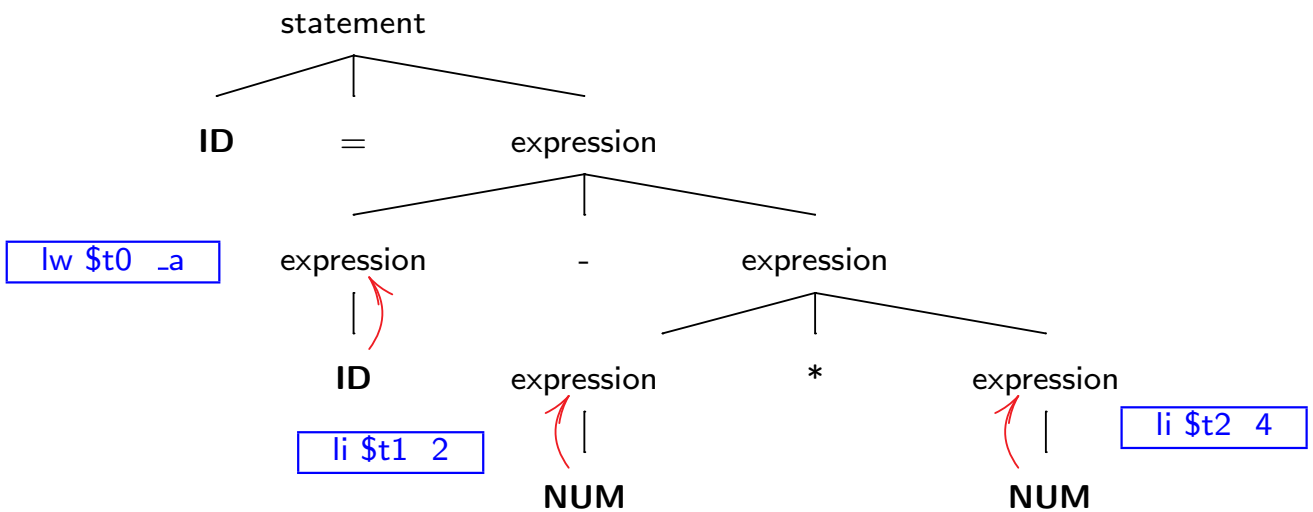
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
| $-E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
| (E_1)	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

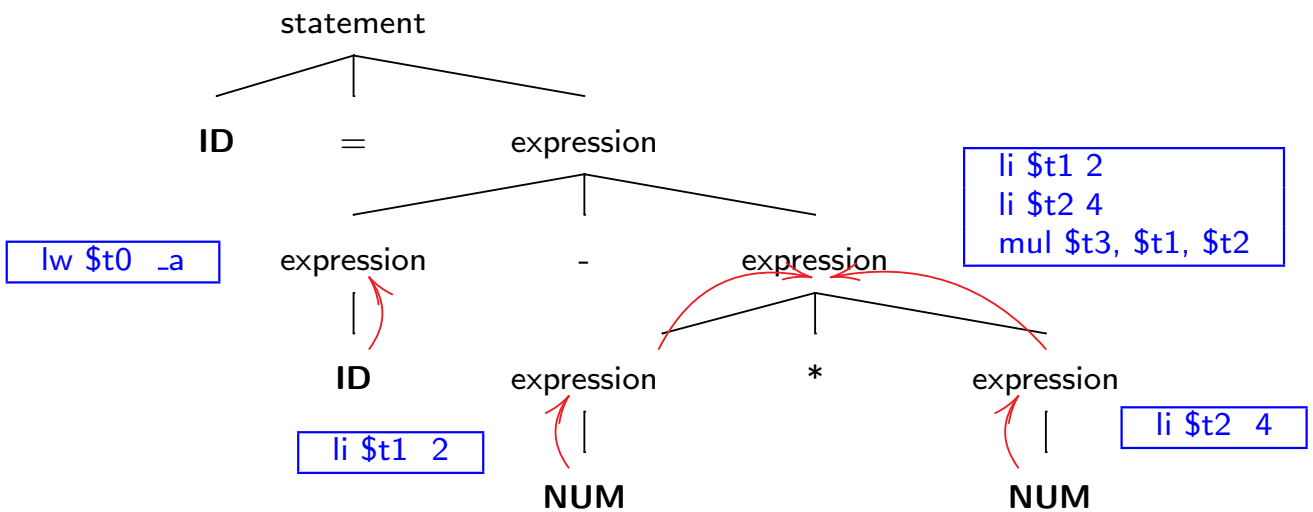
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
$\mid -E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
$\mid (E_1)$	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

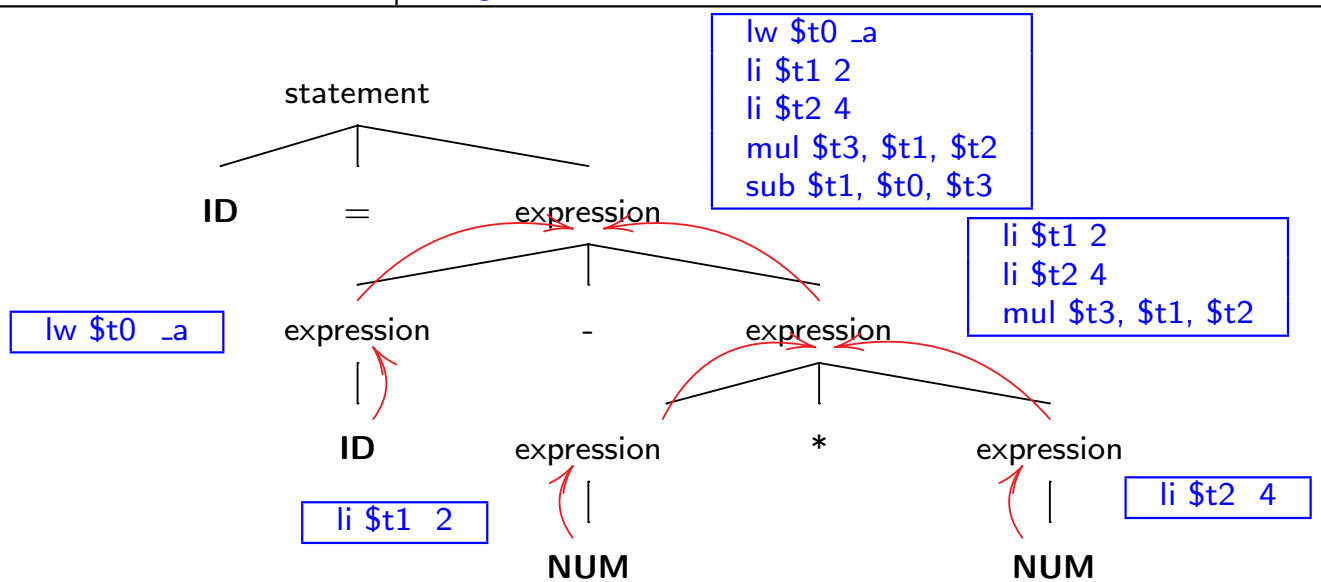
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
| $-E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
| (E_1)	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

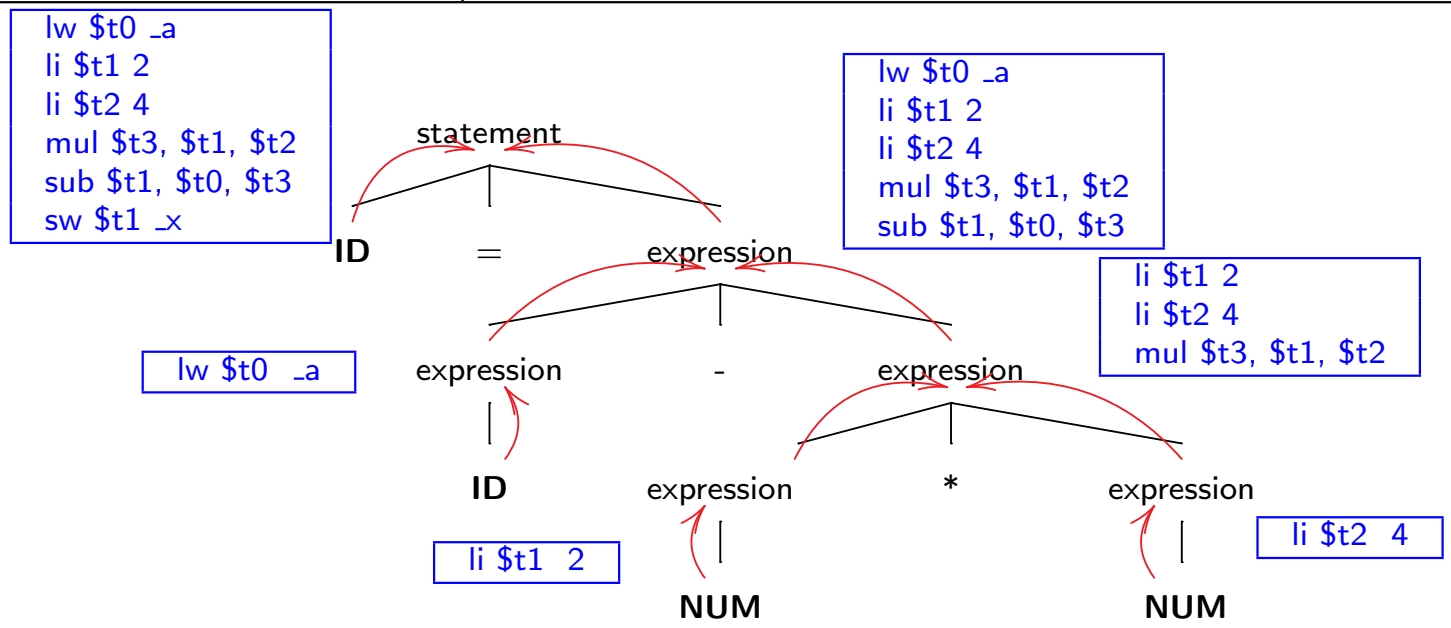
x = a - 2 * 4

REGISTROS:

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

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

Producción	Qué debe realizar el código en C
statement → **ID** = expression	Asignar a $$ una ListaC con las instrucciones de $3 añadiendo instrucción 'sw'
expression → expression + expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'add'
| expression - expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'sub'
| expression * expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'mul'
| expression / expression	Asignar a $$ una ListaC con las instrucciones de $1 y $3 añadiendo instrucción 'div'
| $-E_1$	Asignar a $$ una ListaC con las instruccines de $2 añadiendo instrucción 'neg'
| (E_1)	Asignar a $$ una ListaC con las instrucciones de $2
| **ID**	Asignar a $$ una ListaC con una instrucción 'lw' usando $1
| **NUM**	Asignar a $$ una ListaC con una instrucción 'li' usando $1



SENTENCIA A TRADUCIR:

x = a - 2 * 4

REGISTROS:

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