Projet de compilation

Julien LAVERGNE

Mohamed Amine EL AFRIT

Sommaire

- Phase frontale d'analyse
  - Analyse lexicale
  - Analyse syntaxique
  - Analyse sémantique
- Production du code
- Etude d'un exemple

Analyse lexicale

- But:

  reconnaître les unités lexicales 'lexêmes' du langage.

- Moyen:

  Un analyseur lexical généré par le générateur lexical LEX.

- Exemple:

int	x	=	3	,	y	;
INT	ID	EQ	NUM	VIR	ID	PV

Analyse syntaxique

- But:

  expliciter la structure implicite des instructions en produisant un arbre syntaxique de lexêmes.

- Moyen:

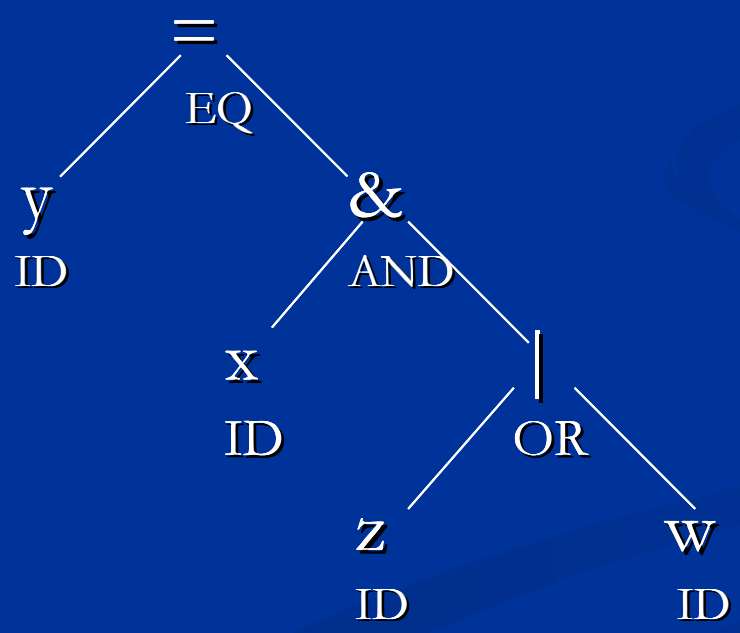
  un analyseur syntaxique généré par le générateur syntaxique YACC

- Exemple

y = x & z | w ;

Priorité:

La priorité entre les opérations ‘&’ et ‘|’ en absence de parenthèses est donnée de gauche à droite.



Exemple

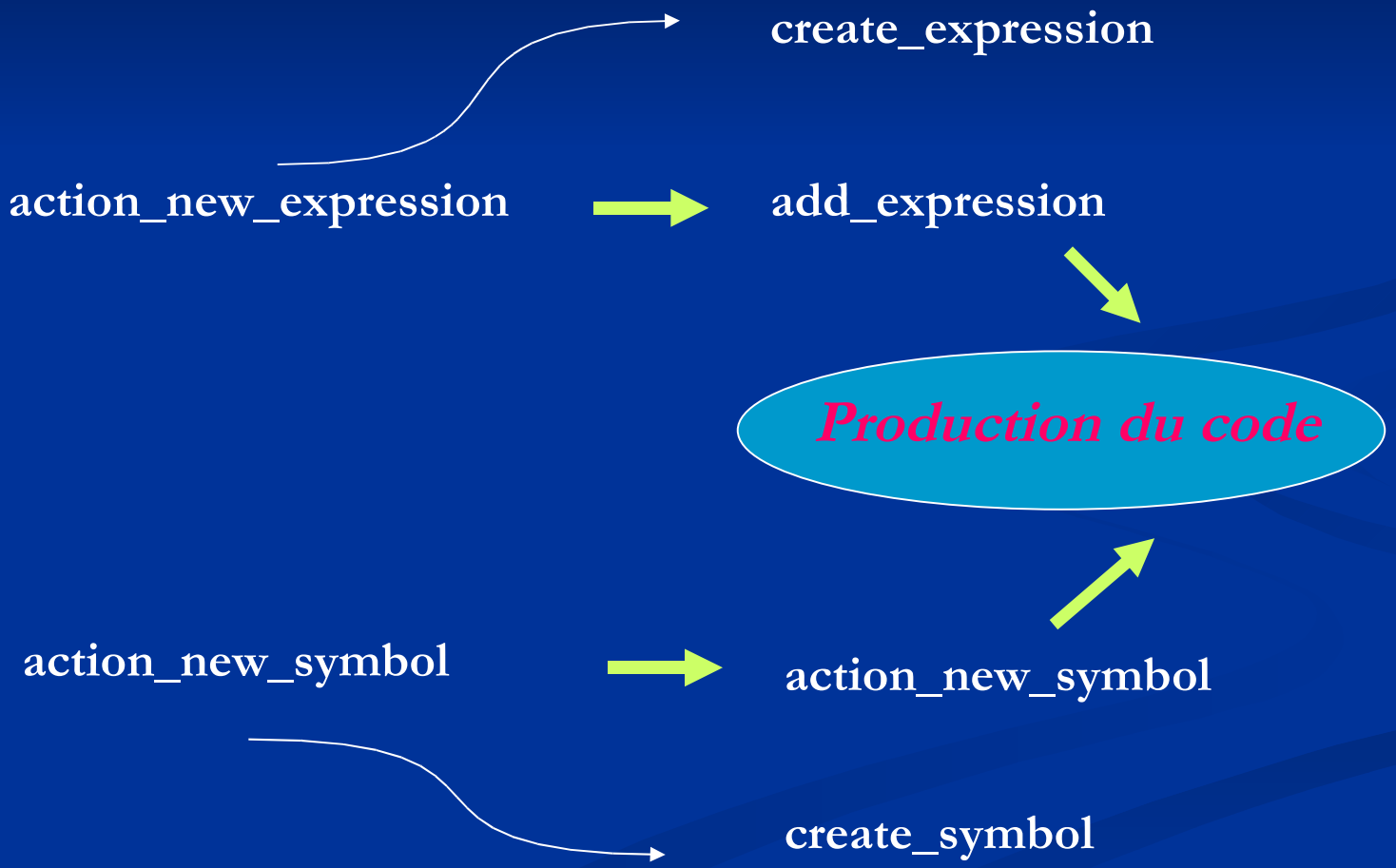
Code source

```
int y = 1, z=2;
int x=3*(y+z);
z=x+y;
```

Code source

```
int y_0;
y_0 = 1;
int z_0;
z_0 = 2;
int var0_0;
var0_0 = y_0 + z_0;
int var1_0;
var1_0 = 3 * var0_0;
int x_0;
x_0 = var1_0;
int var2_0;
var2_0 = x_0 + y_0;
z_0 = var2_0;
```

Actions sémantiques



Analyse sémantique

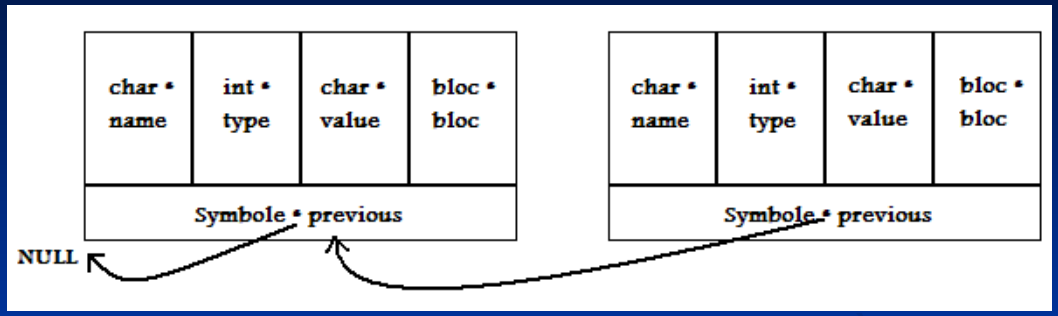
- Analyse des expressions
  - ✓ Table des expressions
- Analyse des symboles
  - ✓ Table des symboles

Analyse des symboles

- But:
  - ✓ vérification des types des opérandes lors des traitements.
  - ✓ Gérer la visibilité des variables et les blocs imbriqués.
- Méthode:
  - ✓ associer à chaque variable son type au sein d'une unité appelée « symbole ».

Définition d'un symbole

```
typedef struct symbol
{
    char* name;
    int type;
    char* value;
    bloc* bloc;
    struct symbol* previous;
} symbol;
```




```
symbol*  search_symbol(char* name){

symbol* s = lastsymbol;

if(   (s->value != NULL) ||
      (s->bloc->level < lastbloclevel) ||
      (s->bloc->id == lastblocid)  ||
      (strcmp(s->name, name) == 0) )

            return s;
}
```

Table des symboles

- La vérification des types
  - ✓ comparaison des champs types des symboles concernés
- Gérer les portées des variables
  - ✓ Chainer les symboles non déclarées au préalable dans le même bloc.
  - ✓ Modifier le **lastSymbol* à chaque rencontre d'un nouveau symbole.

Définition des blocs

```
typedef struct bloc
{
    int id;
    int level;
} bloc;
```

Définition d'une expression

```
typedef struct expression
{
    symbol* result;
    symbol* left;
    int op;
    symbol* right;
    struct expression* previous;
} expression;
```

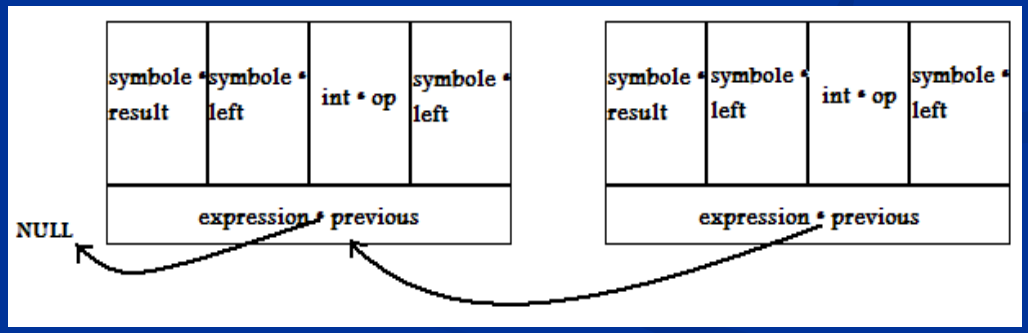


Table des expressions

- Expression de la forme :

$$result = left\ op\ right$$

- Decomposition en 3 @

- Modifier le **lastExpression* à chaque rencontre d'une nouvelle expression.

Production du code 3@

- Définition de variables temporaire pour sauvegarder les calculs intermédiaires.
- Définition des labels pour les structures conditionnelles et les boucles.

Etude d'un exemple

Code source

```
bool x=true, y=false;
float z;
if (x&y) then {
    int x=2;
    z=x*x;
    y=true;
}
else {
    x=y|x;
    x=false;
};
```

Code cible

```
bool x_0;
x_0 = true;
bool y_0;
y_0 = false;
float z_0;
bool var0_0;
var0_0 = x_0 && y_0;
if (!var0_0) then
    GOTO L0;
int x_1;
x_1 = 2;
int var1_1;
var1_1 = x_1 * x_1;
z_0 = var1_1;
y_0 = true;
GOTO L1;
L0;
bool var2_2;
var2_2 = y_0 || x_0;
x_0 = var2_2;
x_0 = false;
L1;
```

Étapes du travail

- Une phase de réflexion.
- Une phase de développement et discussion des problèmes rencontrés.
- Une phase de tests de correction des erreurs et rédaction du rapport en parallèle.

Merci pour votre attention