UFR IM²AG

UNIVERSITÉ Grenoble Alpes

UE INF404 - Projet Logiciel

Projet “Interpréteur” (instruction itérative)

Soutenances et Examen final

L2 Informatique

Année 2023 - 2024

Au menu

Au menu

Objectifs du projet

Ecrire un interpréteur

1. choisir ce que l'on veut interpréter ...
2. définir le langage d'entrée
   alphabet, lexique, syntaxe, sémantique
3. écrire les fonctions d'analyse (lexicale et syntaxique)
4. définir et produire l'Ast
5. écrire le "traitement" de l'Ast

$\Rightarrow$ même démarche que pour la calculette

(et réutilisation partielle possible de certains modules !)

Interpréteur pour un “petit” langage de programmation

1. calculette (L0)

```
12 - 5 * (2+3)
```

2. affectations (L1) [TP5]

```
X = 2 ;
Y = X + 2 ;
```

3. entrées-sorties (L1+) [TP6]

```
lire(X) ;
ecrire (X+12) ;
```

4. instructions conditionnelles (L2) [TP7]

```
si X>0 alors Y = X + 2 sinon Y = 3 fsi ;
```

5. instruction itérative (L3) [TP8]

```
tanque X<10 faire X = X * 2 fait ;
```

Interpréteur pour un “petit” langage de programmation

1. calculette (L0)

```
12 - 5 * (2+3)
```

2. affectations (L1) [TP5]

```
X = 2 ;
Y = X + 2 ;
```

3. entrées-sorties (L1+) [TP6]

```
lire(X) ;
ecrire (X+12) ;
```

4. instructions conditionnelles (L2) [TP7]

```
si X>0 alors Y = X + 2 sinon Y = 3 fsi ;
```

5. instruction itérative (L3) [TP8]

```
tanque X<10 faire X = X * 2 fait ;
```

Interpréteur pour un “petit” langage de programmation

1. calculette (L0)

```
12 - 5 * (2+3)
```

2. affectations (L1) [TP5]

```
X = 2 ;
Y = X + 2 ;
```

3. entrées-sorties (L1+) [TP6]

```
lire(X) ;
ecrire (X+12) ;
```

4. instructions conditionnelles (L2) [TP7]

```
si X>0 alors Y = X + 2 sinon Y = 3 fsi ;
```

5. instruction itérative (L3) [TP8]

```
tanque X<10 faire X = X * 2 fait ;
```

Interpréteur pour un “petit” langage de programmation

1. calculette (L0)

```
12 - 5 * (2+3)
```

2. affectations (L1) [TP5]

```
X = 2 ;
Y = X + 2 ;
```

3. entrées-sorties (L1+) [TP6]

```
lire(X) ;
ecrire (X+12) ;
```

4. instructions conditionnelles (L2) [TP7]

```
si X>0 alors Y = X + 2 sinon Y = 3 fsi ;
```

5. instruction itérative (L3) [TP8]

```
tanque X<10 faire X = X * 2 fait ;
```

Interpréteur pour un "petit" langage de programmation

1. calculette (L0)

```
12 - 5 * (2+3)
```

2. affectations (L1) [TP5]

```
X = 2 ;
Y = X + 2 ;
```

3. entrées-sorties (L1+) [TP6]

```
lire(X) ;
ecrire (X+12) ;
```

4. instructions conditionnelles (L2) [TP7]

```
si X>0 alors Y = X + 2 sinon Y = 3 fsi ;
```

5. instruction itérative (L3) [TP8]

```
tanque X<10 faire X = X * 2 fait ;
```

Au menu

Langage source

- séquence d'instructions
- différentes instructions :
  - affectations
  - lectures/écritures
  - instructions conditionnelles
  - itérations
  - etc.

Exemple

```
lire (X) ;
Y := 1 ;
tanque X > 0 faire
     Y = Y * X ;
     X = X -1 ;
fait ;
ecrire (Y) ;
```

Syntaxe d'un **Programme**

On étend la grammaire : un **programme** est une séquence d'**instructions** (`seq_inst`), où chaque instruction est soit une affectation soit une autre instruction (`lire`, etc.).

Programme = séquence d'Instructions

$$
\begin{aligned}
\text{pgm} &\rightarrow \text{seq\_inst} \\
\text{seq\_inst} &\rightarrow \text{inst suite\_seq\_inst} \\
\text{suite\_seq\_inst} &\rightarrow \texttt{SEPINST seq\_inst} \\
\text{suite\_seq\_inst} &\rightarrow \varepsilon \\
\text{inst} &\rightarrow \texttt{IDF AFF eag} \\
\text{inst} &\rightarrow \texttt{LIRE PARO IDF PARF} \\
\text{inst} &\rightarrow \texttt{ECRIRE PARO eag PARF} \\
\text{inst} &\rightarrow \text{autres formes d'instructions } \ldots
\end{aligned}
$$

Construction d'un Arbre Abstrait (AsT)

construire un AST "complet" du programme

Intérêt :

une seule lecture du fichier $\Rightarrow$ plusieurs traitements possibles

- analyse lexicale et syntaxique complète du fichier
- interprétation = parcours de l'AST
- autres applications possibles :
  - verification des types
  - génération de code assembleur
  - etc.

  $\rightarrow$ par parcours de l'AST ...

Structure de l'Arbre Abstrait ?

```
lire (X) ;
Y := X+1 ;
ecrire (Y * 2) ;
```

3 types d'instructions sur cet exemple :

1. instruction d'affectation (`X := 1`)
2. instruction de lecture (`lire (X)`
3. instruction d'écriture (`ecrire (Y * 2)`

$\Rightarrow$ 4 (nouveaux) types de noeuds dans l'arbre abstrait :

- `N_SEPINST`, séparateur d'instructions (avec 2 fils)
- `N_AFF`, instruction d'affectation (avec 2 fils)
- `N_LIRE`, instruction de lecture (avec 1 seul fils)
- `N_ECRIRE`, instruction d'écriture (avec 1 seul fils)

. . .et des fonctions de construction (`creer_lire()`, etc.)

Structure de l'Arbre Abstrait ?

```
lire (X) ;
Y := X+1 ;
ecrire (Y * 2) ;
```

3 types d'instructions sur cet exemple :

1. instruction d'affectation (`X := 1`)
2. instruction de lecture (`lire (X)`
3. instruction d'écriture (`ecrire (Y * 2)`

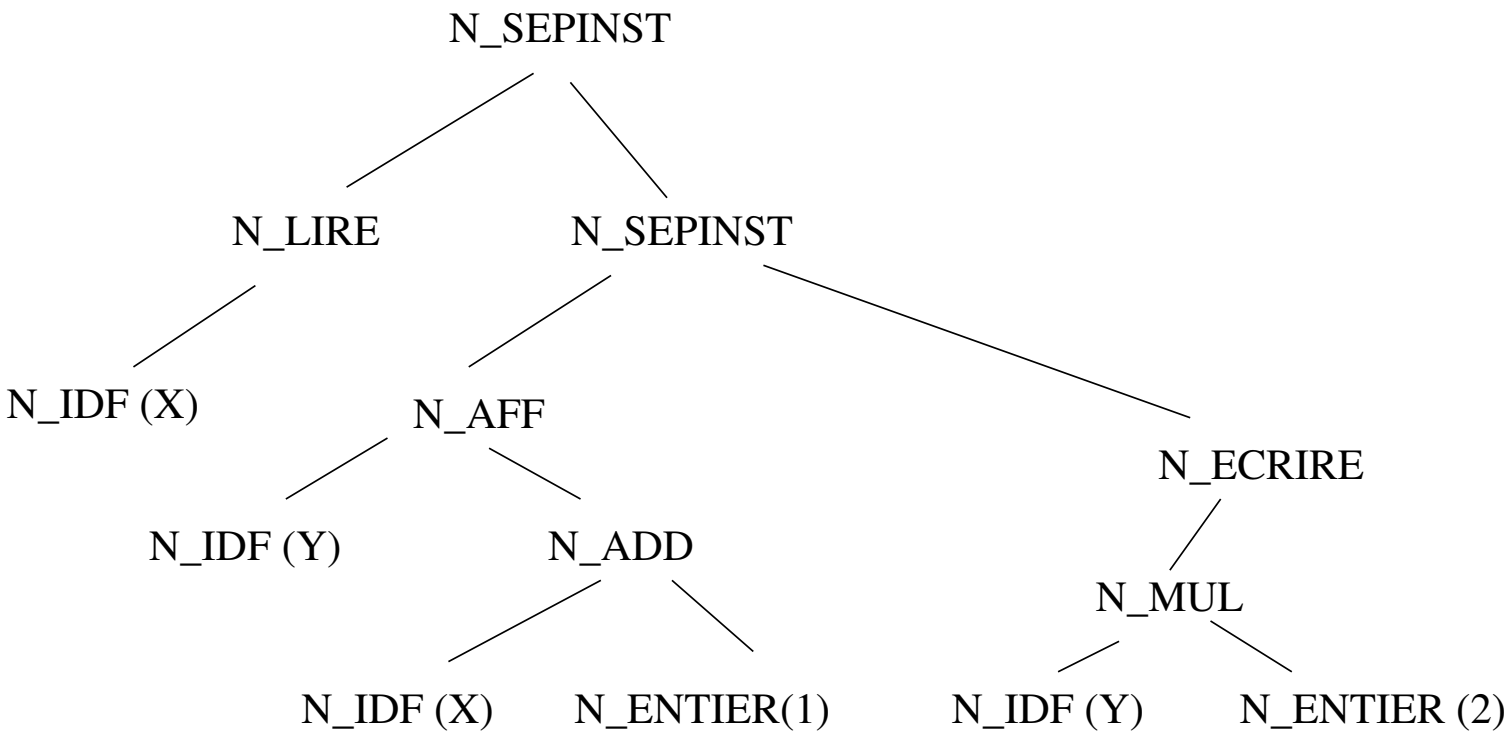
$\Rightarrow$ 4 (nouveaux) types de noeuds dans l'arbre abstrait :

- `N_SEPINST`, séparateur d'instructions (avec 2 fils)
- `N_AFF`, instruction d'affectation (avec 2 fils)
- `N_LIRE`, instruction de lecture (avec 1 seul fils)
- `N_ECRIRE`, instruction d'écriture (avec 1 seul fils)

...et des fonctions de construction (`creer_lire()`, etc.)

Arbre Abstrait de l'exemple précédent

```
lire (X) ;
Y := X+1 ;
ecrire (Y * 2) ;
```



Construction de l'Arbre Abstrait (1)

Etendre l'analyse syntaxique et les modules Ast en ajouter les nouveaux types de noeuds et les procédures de construction associées.

```
Rec_seq_inst (A : resultat Ast) =
  A1 : Ast ;
  Rec_inst (A1)
   // produit l'Ast A1 de l'instruction lue
  Rec_suite_seq_inst (A1, A)
   // produit l'Ast A de la sequence d'instructions lues

Rec_suite_seq_inst (A1 : donné Ast; A : resultat Ast) =
  Ast A2 ;
  selon LC.nature
    cas SEPINST :
      avancer() ;  // on lit le SEPINST
      Rec_seq_inst (A2) ;
      A := creer_seqinst(A1, A2)
     // cree un noeud N_SEPINST de fils gauche A1 et de fils droit A2
   sinon :  // epsilon
     A := A1
```

Construction de l'Arbre Abstrait (2)

inst → IDF AFF *eag*

inst → LIRE PARO IDF PARF

inst → ECRIRE PARO eag PARF

inst → SI condition ALORS seq_inst SINON seq_inst FSI

Traitement par cas selon le type d'instruction

```
Rec_inst (A : resultat Ast)  =
  Ag, Ad : Ast ;
  selon LC().nature
     cas IDF : ... // A := Ast d'une affectation
     cas LIRE : ... // A := Ast d'une lecture
     cas ECRIRE : ... // A := Ast d'une ecriture
     cas SI :  ... // A := Ast d'une inst. conditionnelle
     sinon : Erreur()
```

Interprétation du programme complet

Parcours (récursif) de l'Ast du programme et traitement par cas :

```
interpreter (A : Ast)
  selon A.nature
     cas N_SEPINST :
              interpreter(A.fils_gauche) ;
              interpreter(A.fils_droit) ;
     cas N_AFF : interpreter_aff(A) ;
     cas N_LIRE : interpreter_lire(A) ;
     cas N_ECRIRE : interpreter_ecrire (A) ;
     cas N_IF : interpreter_si_alors_sinon (A) ;
     etc.
     sinon : Erreur() ;


programme principal
    A : Ast ;
    analyser (fichier, A) ; // A contient l'Ast du programme lu
    interpreter(A) ;
    ecrire_TS() ; // on affiche me contenu de la table des symboles
```

Au menu

Définir le langage ?

Exemple

```
lire (X) ;
Y := 1 ;
tanque X > 0 faire
     Y = Y * X ;
     X = X -1
fait ;
ecrire (Y)
```

Que doit-on modifier par rapport à L2 ?

Analyse Lexicale

Nouveaux mots-clés ...

`tanque`, `faire`, `fait`

Reconnaissance des mots-clés (détail)

Dans la procédure `Reconnaitre_Lexeme` :

1. ajouter les lexèmes `TANQUE`, `FAIRE`, `FAIT` au type `Nature_Lexeme`
2. déclarer un tableau de 9 mot-clés (de 20 caractères max)

```
#define NB_MOTCLE 9
char motCle[NB_MOTCLE][20] = {"lire", "ecrire", "si",
    "alors", "sinon", "fsi", "tanque", "faire", "fait"}
```

3. une suite de lettres est considérée (a priori) comme un IDF ...
4. vérifier alors si cet IDF est ou non un mot-clé

```
for (i=0 ; i<NB_MOTCLE ; i++)
    if (strcmp(lexeme_en_cours.chaine, motCle[i]) == 0) {
        switch(i) {
            case 0: lexeme_en_cours.nature = LIRE; break ;
            case 1: lexeme_en_cours.nature = ECRIRE; break ;
                    ...
            case 6: lexeme_en_cours.nature = TANQUE; break ;
                    ...
            default: break ;
        }
    }
```

Analyse Syntaxique (1)

On étend la grammaire : on ajoute l'instruction conditionnelle ...

Programme = séquence d'Instructions

$$
\begin{array}{rcl}
\text{pgm} & \rightarrow & \text{seq\_inst} \\
\text{seq\_inst} & \rightarrow & \text{inst suite\_seq\_inst} \\
\text{suite\_seq\_inst} & \rightarrow & \texttt{SEPINST seq\_inst} \\
\text{suite\_seq\_inst} & \rightarrow & \varepsilon \\
\text{inst} & \rightarrow & \texttt{IDF AFF eag} \\
\text{inst} & \rightarrow & \text{autres formes d'instructions ...} \\
\text{inst} & \rightarrow & \texttt{TANQUE condition FAIRE seq\_inst FAIT}
\end{array}
$$

Analyse Syntaxique (2)

Syntaxe d'une Instruction Conditionnelle

inst $\rightarrow$ `TANQUE` condition `FAIRE` seq_inst `FAIT`

Syntaxe d'une Condition ?

Comme pour l'instruction conditionnelle . . .

expression boolénne "simple"

`X < 3, Y >= 42`

comparaison entre 2 opérandes entiers :

condition $\rightarrow$ eag `OPCOMP` eag

Analyse Syntaxique (2)

Syntaxe d'une Instruction Conditionnelle

inst $\rightarrow$ `TANQUE` condition `FAIRE` seq_inst `FAIT`

Syntaxe d'une Condition ?

Comme pour l'instruction conditionnelle . . .

expression boolénne "simple"

`X < 3, Y >= 42`

comparaison entre 2 opérandes entiers :

condition $\rightarrow$ eag `OPCOMP` eag

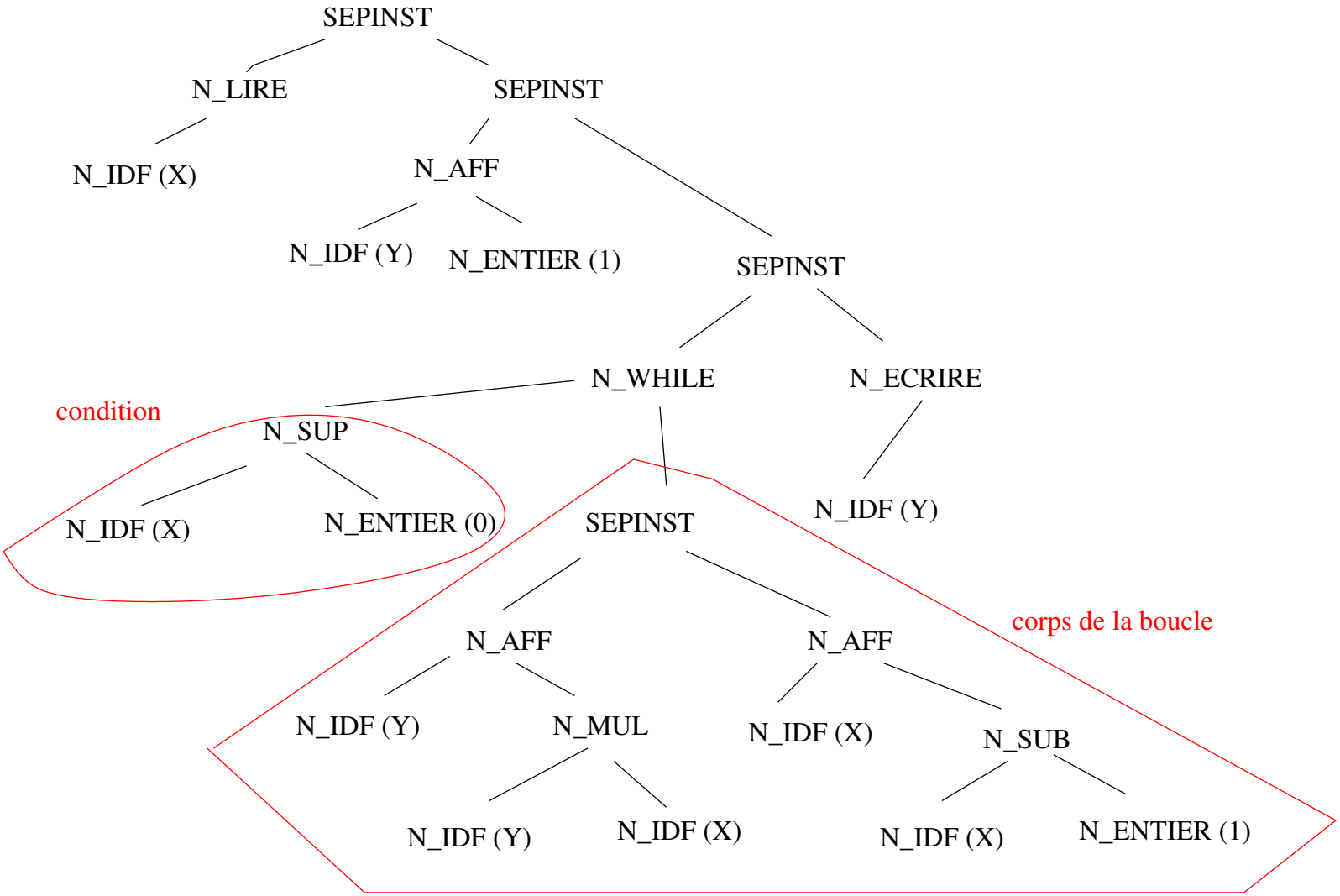
Structure de l'Arbre Abstrait ?

```
  lire (X) ;
  Y := 1 ;
  tanque X > 0 faire
       Y = Y * X ;
       X = X -1
  fait ;
  ecrire (Y)
fsi
```

Quatre instructions sur cet exemple :

1. une instruction de lecture (`lire (X)`)
2. une affectation (`Y := 1`)
3. une instruction itérative :
   - une condition (`X > 0`)
   - un "corps de boucle" avec 2 affectations
4. une instruction d'écriture (`ecrire (Y)`)

Arbre Abstrait de l'exemple précédent



Construction de l'Arbre Abstrait

inst → TANQUE condition FAIRE seq_inst FAIT

⇒ un (dernier !) nouveau type de noeud dans l'arbre abstrait :
N_WHILE, instruction itérative (avec 2 fils)

```
Rec_inst (A : resultat Ast) =
Acond, Abody : Ast
   selon (LC.nature)
         ...
         cas TANQUE:
             avancer
             Rec_condition(Acond)
             si (LC.nature = FAIRE) alors avancer sinon Erreur
             Rec_seq_inst(Abody)
             si (LC.nature = FAIT) alors avancer sinon Erreur
             A = creer_while(Acond, Abody)
                // cree le noeud N_WHILE a partir de ses 2 fils
         ...
```

Construction de l'Arbre Abstrait

inst → `TANQUE` condition `FAIRE` seq_inst `FAIT`

⇒ un (dernier !) nouveau type de noeud dans l'arbre abstrait :
`N_WHILE`, instruction itérative (avec 2 fils)

```
Rec_inst (A : resultat Ast) =
Acond, Abody : Ast
   selon (LC.nature)
         ...
         cas TANQUE:
             avancer
             Rec_condition(Acond)
             si (LC.nature = FAIRE) alors avancer sinon Erreur
             Rec_seq_inst(Abody)
             si (LC.nature = FAIT) alors avancer sinon Erreur
             A = creer_while(Acond, Abody)
                // cree le noeud N_WHILE a partir de ses 2 fils
         ...
```

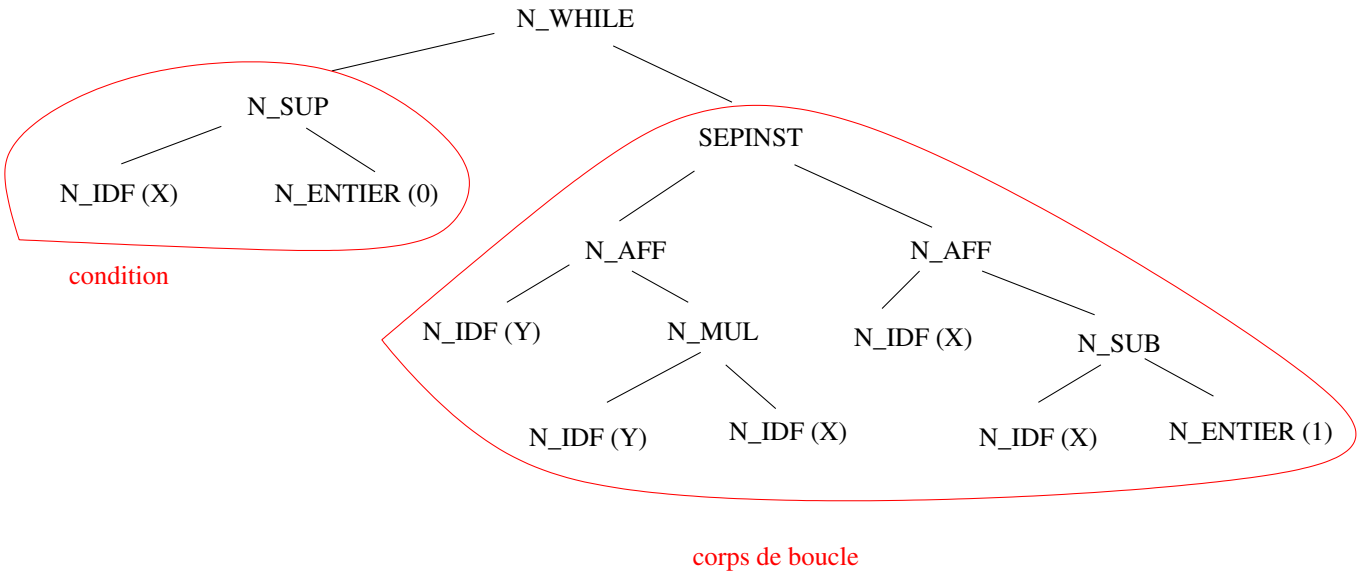
Construction de l'Arbre Abstrait

inst → TANQUE condition FAIRE seq_inst FAIT

⇒ un (dernier !) nouveau type de noeud dans l'arbre abstrait :
N_WHILE, instruction itérative (avec 2 fils)

```
Rec_inst (A : resultat Ast) =
Acond, Abody : Ast
   selon (LC.nature)
         ...
         cas TANQUE:
             avancer
             Rec_condition(Acond)
             si (LC.nature = FAIRE) alors avancer sinon Erreur
             Rec_seq_inst(Abody)
             si (LC.nature = FAIT) alors avancer sinon Erreur
             A = creer_while(Acond, Abody)
                // cree le noeud N_WHILE a partir de ses 2 fils
         ...
```

Interprétation d'une instruction itérative (1)



On exécute la corps de la boucle (le fils gauche) tant que le condition (le fils droit) est vraie . . .

Interprétation d'une instruction itérative (2)

```
interpreter_while (A : Ast)
    // A est un noeud N_WHILE
    tanque valeur_booleenne(A.fils_gauche) faire
       interpreter (A.fils_droit) ;
```

Est-ce que cete itération termine toujours ? Pourquoi ? ?

```
valeur_booleeenne (A : Ast)
    // évalue l'arbre abstrait d'une condition
    valeurg, valeurd : valeurs de fils gauche et droit
    valeurg = evaluer(A.fils_gauche) ;
    valeurd = evaluer(A.fils_droit) ;
    selon A.nature
        cas N_EGAL : return (valeurg = valeur_d)
        cas N_SUP : return (valeurg > valeur_d)
        etc.
```

Interprétation d'une instruction itérative (2)

```
interpreter_while (A : Ast)
    // A est un noeud N_WHILE
    tanque valeur_booleenne(A.fils_gauche) faire
       interpreter (A.fils_droit) ;
```

Est-ce que cete itération termine toujours ? Pourquoi ? ?

```
valeur_booleeenne (A : Ast)
    // évalue l'arbre abstrait d'une condition
    valeurg, valeurd : valeurs de fils gauche et droit
    valeurg = evaluer(A.fils_gauche) ;
    valeurd = evaluer(A.fils_droit) ;
    selon A.nature
        cas N_EGAL : return (valeurg = valeur_d)
        cas N_SUP : return (valeurg > valeur_d)
        etc.
```

Interprétation d'une instruction itérative (2)

```
interpreter_while (A : Ast)
    // A est un noeud N_WHILE
    tanque valeur_booleenne(A.fils_gauche) faire
       interpreter (A.fils_droit) ;
```

Est-ce que cete itération termine toujours ? Pourquoi ? ?

```
valeur_booleeenne (A : Ast)
    // évalue l'arbre abstrait d'une condition
    valeurg, valeurd : valeurs de fils gauche et droit
    valeurg = evaluer(A.fils_gauche) ;
    valeurd = evaluer(A.fils_droit) ;
    selon A.nature
        cas N_EGAL : return (valeurg = valeur_d)
        cas N_SUP : return (valeurg > valeur_d)
        etc.
```

Dans la suite . . .

- continuer les TP5, TP6 et TP7 . . .
  en vous aidant *si nécessaire* du corrigé de la calculette . . .
- poursuivre avec le TP8
- réfléchir à d'autres extensions
  boucle "for" ? procédures ? ?

Au menu

Organisation

- "Inscription" **obligatoire** sur Moodle (groupes du projet)
  . . .avant le 8 avril au soir ! . . .
- Planning élaboré la semaine du 8 avril
- Soutenances les semaines du **15/04** et **29/04**
  (dans les créneaux de cours et TP)

40% de la Note Finale

Contenu

objectif : montrer ce qui tourne . . .

Déroulement

- tous les membres du groupe doivent être présents !
- entre 5 et 10 mn de **démonstration**
  - préparer des exemples !
  - savoir ce que vous voulez montrer sur ces exemples
- 5mn de **questions** (organisation, réalisation, etc.)

Documents à rendre

- document PDF (une page) :
  fonctionnalités, un exemple de “programme” et le résultat obtenu
- code source complet du projet et des exemples
- fichier README (compilation, exécution)

Au menu

Examen Final (écrit)

Rappels

- Durée : 2 heures
- Coefficient : 40% de la Note Finale
- session 1 entre le 15 et le 26 mai

Contenu de l'examen ($\hookrightarrow$ tout ce qui a été vu en cours ...)

- définition d'un langage (lexique, syntaxe, sémantique)
- structure d'un interpréteur
- analyse lexicale (automate finis) et syntaxique (grammaire)
- structures intermédiaires (arbre abstrait, table des symboles, etc.)
  parcours et construction

Annales (sujets et corrigés) disponibles sur Moodle !
Questions possibles par mail ...