TELECOM Nancy 2ème année
Mercredi 10 décembre 2014
Durée : 1h30 mn (14h00 - 15h30)

Examen du module TRAD1

Les documents ne sont pas autorisés.
La note finale tiendra compte du soin apporté à la rédaction de la copie : une copie illisible n'est pas à votre avantage.
Toute réponse écrite au crayon de papier sera effacée par mes soins.
Les réponses non justifiées n'apporteront aucun point à votre note.

EXERCICE 1 : Analyse syntaxique ascendante. (5 points, 20 mn)

Soit la grammaire suivante dans laquelle l'alphabet terminal est constitué des symboles de l'ensemble $\{y, +\}$, et S étant l'axiome.

S	$\longrightarrow$	A	r_0
A	$\longrightarrow$	A + A	r_1
		B + +	r_2
B	$\longrightarrow$	y	r_3

1. Donnez l'arbre syntaxique de l'expression $y + + + y + +$
2. Déroulez l'exécution d'un analyseur ascendant sur cette entrée. Vous présenterez le résultat sous forme d'un tableau à trois colonnes : pile, actions, texte source avec le marqueur de la tête de lecture. Pour répondre à cette question, *on ne vous demande pas de construire un automate LR(1) ni la table d'analyse.*
3. Construisez complètement l'automate LR(1) (8 états, inutile d'augmenter la grammaire). **Ne construisez pas la table d'analyse LR(1).**

EXERCICE 2 : Fonctions sémantiques en analyse syntaxique ascendante. (3 points, 15 mn)

On rappelle la grammaire d'arbres et listes vue en TD :

A	$\longrightarrow$	V
		(A . A)
		(A S)
S	$\longrightarrow$	, A S
		$\wedge$
V	$\longrightarrow$	entier
		nil

On décide maintenant d'associer aux opérateurs `.` et `,` la sémantique suivante :

- pour l'opérateur `.` on reprend la sémantique associée à l'opérateur arithmétique -
- pour l'opérateur `,` on reprend la sémantique associée à l'opérateur arithmétique - mais en le considérant associatif à droite.
- la sémantique associée à `nil` est la valeur `0`

Ainsi, les expressions suivantes sont évaluées comme suit :

- l'expression `((1.2).(3.nil))` est évaluée comme `((1-2) - (3-0))` et vaut `-4`.
- l'expression `(2, 23, 34)` est évaluée comme `(2 -(23 - (34-0)))` et vaut `13`.

Ecrivez les fonctions sémantiques permettant de calculer la valeur associée à une liste. Vous pourrez passer par l'utilisation d'un attribut si vous le souhaitez.

EXERCICE 3 : Arbre abstrait. (4 points, 20 mn)

Soit le programme Java suivant :

```
public class MyString{

 // attributs
 private String content;
 private int nb;

 // constructeur
 public MyString(String content,int i){
    this.content=content;
    nb=i;
 }

 // fonctions
 public String toString(){
    String res="content"+"("+nb+"):="+content+";";
    return res;
 }

 // main
 public static void main(String[] args){
    MyString test=new MyString("test",1);
    System.out.println(test);
 }
}
```

On rappelle qu'un *arbre abstrait* (ou encore AST) est un arbre qui ne garde plus trace des détails de l'analyse syntaxique, c'est à dire qu'il est indépendant de la grammaire, mais il mémorise la structure du programme.

On vous demande de dessiner un AST de ce programme.
Pour cela, la racine pourra être le nom de la classe (MyString), et vous pourrez créer les noeuds ATTR pour les attributs, CONS pour les constructeurs, FUNCTS pour les fonctions et MAIN pour la fonction main (d'autres noeuds seront certainement à définir...) Vous dessinerez ces 4 sous-arbres.

EXERCICE 4 : Questions diverses et mémoire à l'exécution. (5 points, 15 mn)

Réponses courtes et concises attendues.

▷ **Question 1.** On vous énonce que *"Trouver l'appel approprié d'une procédure Q() à partir d'un appel d'une procédure P() est une décision dynamique"* : est-ce vrai ou faux ? (justification en une seule phrase)

▷ **Question 2.** Lors de vos révisions de Trad1, votre binôme vous affirme que *"Trouver la déclaration qui s'applique à une variable non locale x dans une procédure imbriquée P() est une décision statique"* : a-t-il raison ou non ? (justification en une seule phrase)

▷ **Question 3.** Quelles sont, dans l'ordre, les étapes classiques d'une chaîne de compilation ?

▷ **Question 4.** A quel moment le compilateur crée-t-il la Table Des Symboles (TDS) ? Citez 4 exemples d'informations que le compilateur enregistre dans la TDS.

▷ **Question 5.** A quoi correspond le déplacement associé à une variable dans la TDS ? Où cela sert-il ?

▷ **Question 6.** Lors de l'élaboration d'un AST, quelles différentes actions peut-on entreprendre lors du traitement des terminaux de la grammaire initiale (c'est à dire, où peuvent-ils apparaître dans l'AST) ?

▷ **Question 7.** Le fragment de code ci-dessous propose un code C qui calcule récursivement les nombres de Fibonacci.

```
    int f(int n) {
        int t, s;
        if (n < 2) return 1;
        s = f(n - 1);
        t = f(n - 2);
        return s+t;
    }
```

Supposons qu'un bloc d'activation (ou enregistrement dans la pile) pour f comporte, dans l'ordre : valeur de retour (et non pas adresse de retour...), paramètre n, variable locale s, variable locale t. Il y a, bien évidemment, d'autres éléments dans le bloc d'activation, mais inutiles pour cette question.
On supposera dans cet exercice que l'appel initial est *f(5)*.

a- Dessinez complètement l'arbre des appels pour *f(5)*.

b- Quel inconvénient voyez-vous à cette implémentation de l'algorithme de calcul des nombres de Fibonacci ?

c- A quoi ressemble la pile et ses blocs d'activation juste avant le cinquième retour de *f(1)* ?

EXERCICE 5 : Pile à l'exécution. (3 points, 10 mn)

Soit le programme suivant écrit en langage C (le langage standard ne supporte pas cette construction mais certains compilateurs l'acceptent, tel `gcc` par exemple). Comme dans les exemples vus en cours, la portée de la fonction imbriquée est limitée par la fonction englobante. Ce programme est correct, il se compile et s'exécute sans erreur.

```
#include <stdio.h>
int x = 0;   int y = 10;

typedef struct{
   int A1;
   float A2;
   int A3;
   } EX;

  void P (EX z, int *y) {

     int Q (EX a, int *y) {
          a.A3 = a.A3 * 3;
          *y = *y * 10;
          printf("Point 1 %d\n", x + a.A3 + *y);
             --> Pile à dessiner ici : POINT 1
          return(x + a.A3 + *y);
     }

    void S (int x) {
      z.A1 = z.A1 * 2;
      *y = *y * 3;
      printf("Point 2: y = %d\n", *y);
      x = z.A1 - (Q(z, y) - x);
      printf("Point 4 : x = %d \n",  x);
     }

    x = *y + 1;
    printf("Point 3: x = %d\n", x);
    S(x);
   }

int main (void) {
  EX a;   int x;
  x = 50;
  a.A1 = 10; a.A2 = 20.5; a.A3 = 30;
  P(a, &y);
  return 0;
}
```

On suppose que les environnements à l'exécution sont gérés par l'intermédiaire des chaînages dynamique et statique. Les valeurs de retour des fonctions seront rangées dans un registre.

Simulez l'exécution du programme et dessinez la pile au `POINT 1` cité dans le programme. Indiquez également ce qu'il imprime, et dans l'ordre d'exécution.