Examen de Compilation
première session 2023

L'épreuve dure 3h et les documents ne sont pas autorisés.
L'examen est sur 20+2 points. Les deux points suplémentaires sont en bonus.

Exercice 1 (cours, 4pt, 15min)**.** Répondez à la question suivante (moins de 2 lignes) :

1. Quelle est la diférence entre un compilateur et un interpreteur
2. Qu'attend-on en entrée du générateur de lexeur ?
3. Qu'attend-on en sortie du parseur ?
4. Qu'attend-on en entrée d'une machine virtuelle ?

Répondez aux la questions suivantes (3-4 lignes par réponses, pas plus) :

5. Que sont les contextes ?
6. Quelle sont les différences entre SLR, LR0, LR1 et LL ?

Exercice 2 (cours+prog+parseur, 5pt, 60min)**.** Dans le langage de votre choix, écrivez

1. une structure de donnée représentant les grammaires,
2. un fichier de génération de parseur permetant de créer un parseur qui lit une grammaire,[1]
3. deux fonctions calculant les firsts et les follows.

Vous pouvez supposer que vous avez un lexeur qui fournit les tokens de votre choix. On ne sera pas exigeant sur la syntaxe exacte.

Exercice 3 (lexeur, 4pt, 30min)**.** Construir un lexeur reconnaissant les tokens suivants (dans cet ordre de priorité) :

- Les angles en degré, min et sec où chacun peu apparaître ou ne pas apparaître sauf si on a degré et secondes, auquel cas on aura des minutes. Exemples : `23°45'8''`, `5'27''`, `12'` ou `42°2'` mais on ne peut pas avoir `4°18''`.
- Les mots contenant lettres, chiffres, tirets et appostrophe. On peut commencer par une lettre ou un chiffre mais pas une apostrophe; on peut finir par n'importe quoi sauf un tiret; on ne peut pas avoir deux tirets ni deux appostrophes à la suite.

Les commentaires unilignes commançant par \\ doivent être ajouter comme séparateurs

[1] Attention au niveau d'abstration : il s'agit bien d'une grammaire pour lire les grammaires

Exercice 4 (Parser, 3pt, 20min)**.** Donnez le parseur LR0 associé à la grammaire suivante

$$\begin{array}{lll} S & ::= & Ta \mid aTb \\ T & ::= & aSa \mid b \end{array}$$

Exercice 5 (Decompilation, 6pt, 40mn)**.** Voici un programme dans l'assembleur vu en cours. Décompilez-le, c'est à dire essayez de retrouver le programme dont il est originel. Vous pouvez ne décompiler que des parties ou laisser des trous, donnez bien les lignes corespondantes aux parties traduites.
La zone avant le commentaire l.41 est particulièrement dificile, nous vous conseillons de commencer par traduir ce qui vient après le commentaire, d'autant que c'est là que vous aurrez le plus de points.

```
1   NewClo f
2   DecArg x
3   SetVar f
4   NewObj
5   NewObj
6   NewClo c
7   SetObj constructor
8   NewClo g
9   DecArg x
10  SetObj f
11  SetObj prototype
12  SetObj foo
13  NewObj
14  GetVar Bar
15  FrmPrt
16  NewClo d
17  DecVar foo
18  SetObj constructor
19  SetObj prototype
20  SetVar bar
21  GetVar bar
22  GetObj prototype
23  FrmPrt
24  Copy
25  GetObj constructor
26  StCall
27  Swap
28  SetIn  this
29  GetVar foo
30  GetObj prototype
31  FrmPrt
32  Copy
33  GetObj constructor
34  StCall
35  Swap
36  SetIn  this
37  Call
38  SetArg
39  Call
40  #
41  # + simple
42  CsteNb 42
43  SetVar z
44  getVar f
45  StCall
46  GetVar z
47  SetArg
48  Call
49  SetVar z
50  GetVar z
51  CsteNb 100
52  GrStNb
53  ConJmp 15
54  GetVar y
55  Copy
56  GetObj f
57  StCall
58  Swap
59  SetIn  this
60  GetVar f
61  StCall
62  GetVar z
63  SetArg
64  Call
65  SetArg
66  Call
67  Drop
68  Jump   11
69  GetVar y
70  GetObj foo
71    GetVar f
72    Copy
73    StCall
74    Swap
75    SetIn  this
76    GetVar z
77    SetArg
78    Call
79    Drop
80    Halt
81  c GetVar this
82    CsteNb 42
83    SetObj u
84    Return
85  g DecVar y
86    GetVar this
87    GetObj u
88    SetVar y
89    GetVar this
90    GetVar x
91    SetObj u
92    GetVar y
93    Return
94  d GetVar this
95    GetPrt
96    GetObj prototype
97    StCall
98    GetVar this
99    SetIn  this
100   Call
101   Return
102 f GetVar z
103   GetVar x
104   AddiNb
105   Return
```

Instruction	sémantique	pile avant	pile après
AddiNb, SubsNb, MultNb, DiviNb	Push(Pop $\odot_f$ Pop); avec $\odot$ représentant, respectivement, $+$, $-$, $*$ et $/$	n_1::n_2::pile	n_3::pile
GrStNb	Push(Pop $>_f$ Pop);	n_1::n_2::pile	b::pile
CsteNb x	Push(x);	pile	n::pile
Copy	Push(Pull);	v::pile	v::v::pile
Drop	Pop();	v::pile	pile
GetVar n	Push(Get(n))	pile	v::pile
SetVar n	Set(n, Pop())	v::pile	pile
DclVar n	Insert(n, undefind)	pile	pile
NewClo off	Push(NewCloture{cont := CopyCont, code := PC+off+1})	pile	l::pile
Jump offset	PC := PC + off + 1;	pile	pile
ConJmp offset	if Pop then PC := PC + 1; else PC := PC + off + 1;	b::pile	pile
DclArg n	Pull.args.Push(n)	l::pile	l::pile
StCall	Pull.setContext(NewContext(CC))	l::pile	l::pile
SetArg	v := Pop; clot := Pull; n := clot.args.Pop; clot.cont.Insert(n, v);	v::l::pile	l::pile
Call	clot := Pop; Push(NewContinuation{cont := CC, code := PC}) CC := clot.cont; PC := clot.code	l::pile	t::pile
Return	res := Pop; continue := Pop; CC := continue.cont; PC := continue.code; Push(res)	v::t::pile	v::pile
NewObj nom	Push(new Object)	pile	o::pile
GetObj nom	Push(Pop.Get(nom))	o::pile	v::pile
SetObj nom	Pop.Set(nom, Pop.Get)	v::o::pile	pile
FrmPrt	Push(NewObjet{__proto__ = pop()})	o::pile	o::pile