Examen de Compilation
première session 2023

L'épreuve dure 3h et les documents ne sont pas autorisés.

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

1. Donnez une différence entre un clôture et une continuation
2. Qu'attend-on en entrée du générateur des analyse sémentiques ?
3. Qu'attend-on en sortie du lexeur ?
4. Qu'attend-on en entrée du générateur de parser ?

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

5. Expliquez une des façon d'implémenter les exceptions.
6. Qu'est-ce que le "hoisting" et comment le faire dans le compilateur.
7. **Licence uniquement:** Donnez le code assembleur de `x+3` où x peut être un booléen, un entier ou une string.

Exercice 2 (cours+prog+parseur, 4pt, 40min)**.** Dans le langage de votre choix,

1. écrivez une structure de donnée[1] d'AST pour la grammaire suivante
   $\langle com \rangle$ ::= $\langle VAR \rangle = \langle expr \rangle$; | if ($\langle expr \rangle$) $\langle com \rangle$ else $\langle com \rangle$
   $\langle expr \rangle$ ::= $\langle VAR \rangle$ | $\langle INT \rangle$ | $\langle expr \rangle + \langle expr \rangle$
2. une fonction optimisant l'AST en précalculant les tout ce qui ne contient pas de variable.

Exercice 3 (lexeur, 3pt, 20min)**.** Construir un lexeur reconnaissant les tokens suivants (dans cet ordre de priorité) :

- les déclarations de champs de la forme "`toto:`" ou "`foo :`" constitués de lettres minuscules et avec un nombre arbitraires d'espaces avant le ":";
- les iddentifiants qui peuvent être composés de lettres ou de chiffres mais on ne peut pas avoir de lettre après les chiffres. Ex : "`toto42`", "`fooBar`".

Les retours à la ligne (linux et windows) sont utilisés comme séparateurs.

Exercice 4 (Parser, 4(+1)pt, 20(+15)min)**.**

- Donnez le parseur LR0 (et ses conflits) associé à la grammaire suivante
  S ::= Aa | bABb
  A ::= ϵ | bABb
  B ::= Sc | a
- **Masters uniquement:** Donnez le parser SLR (et ses conflits).

[1] struct, classe java ou type OCaml

Exercice 5 (Execution, 4pt, 60mn)**.** Voici un programme dans l'assembleur vu en cours; remplacez "*" par le dernier chiffre de votre n.étudiant. L' instruction `Log` copie la tête de pile (sans la retirer) dans une trace. Donnez la trace écrite par les `Log` pendant l'exécution du programme (et uniquement cette trace). Conseil: il y a environ 200 étapes de calcule à faire, avec de moins en moins de points et de plus en plus de risque d'erreur, n'allez au bout que si vous avez compris le programme.

```
1  CsteNb *
2  Log
3  Copy
4  CsteNb 5
5  GrEqNb
6  CondJmp 3
7  CstNb 5
8  SubiNb
9  Jump -7
10 CstNb 1
11 AddiNb
12 Log
13 CsteNb 7
14 Log
15 CstNb 42
16 CstNb 38
17 NewClot 33
18 DecArg x
19 SetVar f
20 SubiNb
21 Log
22 SubiNb
23 Log
24 SetVar y
25 Local
26 DecVar y
27 CsteNb 3
28 SetVar y
29 u: GetVar y
30 NbToBo
31 ConJmp 39
32 Copy
33 Catch 8
34 Swap
35 GetVar f
36 StCall
37 Swap
38 SetArg
39 Call
40 Swap
41 Drop
42 Log
43 Drop
44 CsteNb 1
45 SubiNb
46 GetVar y
47 CstNb 1
48 SubiNb
49 SetVar y
50 Jump u
51 GetVar x
52 Log
53 NbToBo
54 CndJmp 14
55 CsteNb 0
56 GetVar x
57 GrEqNb
58 CndJmp 2
59 CsteBo false
60 Throw
61 GetVar f
62 StCall
63 GetVar x
64 GetVar y
65 SubiNb
66 SetArg
67 Call
68 Return
69 CsteBo true
70 Return
71 Halt
```

Instruction	sémantique	pile avant	pile après
Concat	Push(PopPop)	s::s::p	s::p
GrEqNb	Push(Pop $\geq_{\mathtt{f}}$ Pop);	n_1::n_2::p	b::p
Copy	Push(Pull);	v::p	v::v::p
Swap	a = pop; b = pop; Push(a); Push(b);	v_1::v_2::p	v_2::v_1::p
Drop	Pop();	v::p	p
GetVar n	Push(Get(n))	p	v::p
SetVar n	Set(n, Pop())	v::p	p
DclVar n	Insert(n, undefind)	p	p
NewClo off	Push(NewCloture{cont := CopyCont, code := PC+off+1})	p	l::p
Jump off	PC := PC + off + 1;	p	p
ConJmp off	if Pop then PC := PC + 1; else PC := PC + off + 1;	b::p	p
DclArg n	Pull.args.Push(n)	l::p	l::p
StCall	Pull.setContext(NewContext(CC))	l::p	l::p
SetArg	v := Pop; clot := Pull; n := clot.args.Pop; clot.cont.Insert(n, v);	v::l::p	l::p
Call	clot := Pop; Push(NewContinuation{cont := CC, code := PC}) CC := clot.cont; PC := clot.code	l::p	t::p
Return	res := Pop; cont := Pop; CC := cont.cont; PC := cont.code; Push(res)	v::t::p	v::p
Local off	CC = NewContext(CC)	p	p
Catch off	Push(NewContinuation{cont = CC, code = PC + off + 1, err = true})	p	t::p
Throw	res := Pop; do c := Pop;while (Type(c) $\neq$ Continuation|| not c.err) CC := c.cont; PC := c.code; Push(res)		