Examen de Compilation
Seconde session 2022

Exercice 1 (cours+réflexion)**.**

1. Java est-il compilé ? interprété ? décrivez succinctement son schéma de compilation (un dessin suffit).
2. Voici quelques vrais et fausses étapes de la génération de parseur. Sélectionnez les vrais étapes et remettez dans l'ordre d'exécution :
   - Élimination des conflits Shift/Action,
   - Élimination des conflits Shift/Shift,
   - Psedo-déterminisation,
   - Écriture de la grammaire,
   - Construction de de l'automate,
   - Lecture de la grammaire.
3. Expliquez, en 4-5 lignes, un mécanisme de compilation des exception (il y en a plusieurs dans le cours, choisissez celui que vous souhaitez).

Exercice 2 (Code)**.**
Définissez des types pour vos AST et écrivez un fichier de génération de parseur dans le langage de votre choix (bison, ocamlyacc, javacc, ect...) dont les actions génèrent un AST et qui reconnaisse un fragment de javascript contenant :

- les expressions composées d'entiers, de Booléens, sommes, inférieur stricte et appels de fonctions,
- les commandes if_then_else et try_catch_,
- les déclarations de fonctions.

L'AST doit aussi contenir le type JavaScript lorsque vous arrivez à le deviner facilement.

On sera très laxiste sur la syntaxe exacte, les bibliothèques et la gestion des erreurs, par contre la structure devra être respectée et les opérateur utilisés doivent exister (avec potentiellement une syntaxe légèrement différente).

Exercice 3 (parseur)**.**
Voici une grammaire dont les terminaux sont a, b et c
$S := SaS \mid R$
$R := Rb \mid Tc \mid \epsilon$
$T := SaT$

1. Cette grammaire est ambiguë, montrez-le.
2. On considérera que l'opérateur binaire a est associatif à gauche et moins prioritaire que l'opérateur unair suffixe c, écrire le parseur SLR correspondant.

Exercice 4 (lexeur)**.** Écrivez des expressions régulières et un lexeur complet reconnaissant les lexèmes suivants :

- les listes de mots entre chevrons (par exemple `<>`, `<foo>` ou `<foo,bar>`).
- les symboles de comparaison (`<=`, `<`, `==`...)
- les identifiants commençants par une minuscule et ne contenant que des minuscules et des chiffres.

Exercice 5. Voici un extrait de programme dans l'assembleur vu en cours, décompilez-le, c'est à dire retrouvez le programme dont il est issu. Si vous n'arrivez pas à tout traduire, indiquez bien les lignes de début et de fin du code traduit.

```
1  NewClot 38
2  DecArg x
3  DecArg y
4  SetVar f
5  CsteNb 32
6  CsteNb 20
7  GetVar n
8  AddiNb
9  SubsNb
10 SetVar n
11 SetVar m
12 GetVar A
13 GetObj prototype
14 FrmPrt
15 GetObj constructor
16 StCall
17 Call
18 SetVar toto
19 GetVar f
20 StCall
21 GetVar m
22 SetArg
23 CsteNb False
24 SetArg
25 Call
26 ConJmp 12
27 GetVar toto
28 GetObj b
29 ConJmp 6
30 GetVar toto
31 GetObj g
32 StCall
33 Call
34 SetVar n
35 Jump 2
36 CstNb 0
37 SetVar n
38 Jump -20
39 Halt
40 GetVar x
41 TypeOf
42 Case 2
43 BoToNb
44 Jump 3
45 Jump 2
46 Noop
47 CstNb nan
48 GetVar x
49 TypeOf
50 Case 2
51 BoToNb
52 Jump 3
53 Jump 2
54 Noop
55 CstNb nan
56 AddiNb
57 Return
```

Instruction	sémantique	pile avant	pile après
Log	Print(Pull);	X:pile	X:pile
AddiNb, SubsNb, MultNb, DiviNb	Push(Pop $\odot_f$ Pop); avec $\odot$ représentant, respectivement, $+$, $-$, $*$ et $/$	#n:#n:pile	#n:pile
LoStNb	Push(Pop $<_f$ Pop);	#n:#n:pile	#b:pile
CstNb x	Push(x);	pile	#f:pile
Copy	Push(Pull);	X:pile	X:X:pile
Swap	échange les 2 premières valeurs de la pile	X:Y:pile	Y:X:pile
GetVar n	Push(Get(n))	pile	#v: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
StCal	Pull.setContext(NewContext(CC))	#l:pile	#l:pile
SetArg	v := Pop; clot := Pull; n := clot.args.Pop; clot.cont.Insert(n, v);	#v:#c:pile	#c: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)	X:#t:pile	X:pile
BoToNb	if Pop then Push(1) else Push(0)	#b:pile	#n:pile
NbToBo	Push(Pop $\neq_f$ 0)	#n:pile	#b:pile
Case p	PC := PC + p * Floor(Pop) + 1;	#n:pile	pile
TypeOf	Peak une valeur, rend 0 sur un Boolean, 1 sur un Number, 2 sur un Undefined, ...	X:pile	#n:X:pile
GetObj nom	Push(Pop.Get(nom))	#o:pile	#v:pile
FrmPrt	Push(NewObjet{__proto__ = pop()})	#o:pile	#o:pile
Noop	Ne fait rien	#pile	#pile
Halt	Arrête la machine		