Examen de Compilation
première session 2021

Exercice 1. En quelques lignes, Décrivez ce qu'est l'analyse sémentique.

Exercice 2. Écrivez un fichier de génération de lexeur dans le langage de votre choix (lex, ocamllex, javacc, ect...) qui :

- lise les entiers et les mettent dans un Token `INT`; on acceptera des entiers commençant par des 0 innutiles,
- lise les chaines de lettres (majuscules et minuscules) et tiret et les mettent dans un token `IDEN`, attention, les mots-clés ci-dessous ainsi qui votre prénom[1] ne doivent pas former des tokens `Iden`
- lise les mots-clés `if`, `else` et `while` en les insérant dans un token `KEYWORD`,
- élimine les espaces et des retours à la ligne sans fournir de token (séparateurs).

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 potentielement une syntaxe légèrement diférente).

Exercice 3. Voici une grammaire dont les terminaux sont a, b et c
$S := EaG \mid Ea$
$E := b \mid FaF$
$F := Ec|\epsilon$
$G := FcS$
Les questions suivantes peuvent être traiter dans l'ordre que vous le souhaitez. Indiquez simplement le numéro avant chaque réponse.

1. Quels sont les non-terminaux ?
2. *bcaacba* est-il un mot de cette grammaire ? Montrez-le.
3. Construire le parseur LR0.
4. Iddentifiez les conflits du LR0.
5. Calculez les firsts et follow.
6. Constrire le parseur SLR.
7. Iddentifiez les conflits.
8. Ceux-ci peuvent-il se résoudre en rajoutant des priorités/associativités. Justifiez.

[1] Si votre prénom est composé, prenez sa première partie, s'il est trop long prenez les 6 premières lettres.

Exercice 4. Voici un programme dans l'assembleur vu en cours; dans la troisième ligne, veuillez insérer votre numéro étudiant à l'endroit indiqué. On utilise une instruction suplémentaire, `Log` qui écrit la tête de pile, sans la retirer, dans une trace. Donnez la trace écrite par les `Log` pendant l'exécution du programme.

```
CsteNb 2000
CsteNb -10000
CsteNb <votre numéro étudiant>
CsteNb 1000
DiviNb
AddiNb
Copy
log
SetVar y
NewClo 28
DecArg y
DecArg x
SetVar f
LoStNb
CsteNb -1000
```

```
Swap
GetVar y
Swap
ConJmp 2
AddiNb
CsteNb -1000
AddiNb
log
GetVar f
StCall
Swap
GetVar f
StCall
Swap
SetArg
CsteNb 100
```

```
SetArg
Call
SetArg
CsteNb 10
SetArg
Call
Halt
GetVar x
GetVar y
SubsNb
Copy
Log
GetVar y
LoStNb
ConJmp -7
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
Halt	Arrête la machine		