Ratrapage de Compilation
première session 2023

Exercice 1 (cours, 3.5pt, 15min)**.** Répondez à la question suivante (moins de 2 lignes) :

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

Répondez à la question suivante (4-5 lignes par réponses, pas plus) :

6. Comment propose-t-on d'implémenter les contextes dans le cours ?

Exercice 2 (Lexeur+prog, 4pt, 30min)**.** Écrivez un fichier de *g*énération de lexeur dans le langage de votre choix (lex, ocamllex, javacc, ect...) qui :

- lise les flotants en écriture scientifique (avec le 'e'), les négatifs, les exposants négatifs, et les flottants spéciaux (infinits, nans...) ne sont pas demandés,
- lise les chaines de lettres (majuscules et minuscules) et tiret et les mettent dans un token `IDEN`, attention, votre prénom[1] ne doit pas former des tokens `Iden`
- lise les opérateurs suivants : le '.' des objets, le '+' des flottants et le '++' pour l'incrémentation.
- (dificile) les changements d'indentation lors des retours à la ligne fournissent des tokens de crochets implicites ouvrants ou fermant, comme en Python.

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 (assembleur+prog, 5pt, 30min)**.** Dans le langage de votre choix (C,Java,OCaml...) écrire une structure de donnée pour l'AST de la grammaire suivante, ainsi que le programme de génération du code assembler minijs-machine associé :

$$
\begin{array}{rcl}
\langle \text{expr} \rangle & := & \langle \text{BOOL} \rangle \\
 & & \langle \text{expr} \rangle == \langle \text{expr} \rangle \\
\langle \text{com} \rangle & := & \text{if } (\langle \text{expr} \rangle) \ \langle \text{com} \rangle \text{ else } \langle \text{com} \rangle \\
 & & \langle \text{expr} \rangle \ ;
\end{array}
$$

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).

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

Exercice 4 (parseur, 5pt, 30min)**.** Voici une grammaire dont les terminaux sont a, b et c

$S := EaE \mid ES$

$E := b \mid EaE$

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. *babbbab* est-il un mot de cette grammaire ? Montrez-le.
3. Construire le parseur LR0.
4. Iddentifiez les conflits du LR0.
5. Ceux-ci peuvent-il se résoudre en rajoutant des priorités/associativités. Justifiez.

Exercice 5 (assembleur, 3pt, 10min)**.** Écrire un programme assembleur minijs-machine représenant le programme suivant

```
try
  if f(x) z=x+1;
catch (e) z=e+1;
z+z;
```

On va supposer que z est forcément un entier et que e peut être un booléen ou un entier.

Instruction	sémantique	pile avant	pile après
AddiNb, SubsNb, MultNb, DiviNb	`Push(Pop` $\odot_{\texttt{f}}$ `Pop)`; avec $\odot$ représentant, respectivement, $+$, $-$, $*$ et $/$	`#n:#n:pile`	`#n:pile`
Equals	`Push(Pop == Pop);`	`#v:#v:pile`	`#v: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`
StCall	`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`
Catch off	`Push(NewContinuation{cont = CC,` `code = PC + off + 1,` `err = true})`	`pile`	t : `pile`
Halt	Arrête la machine		