+1/1/60+

**Vérifiez votre énoncé: les 6 entêtes doivent être +1/1/*xx*+... +1/6/*xx*+.**

# ESIEE Construction des Compilateurs — Sans document ni machine

**Noircir les cases plutôt que cocher.** Renseigner les champs d'identité. Les questions marquées du symbole ♣ peuvent avoir plusieurs réponses justes. Toutes les autres questions n'ont qu'une seule réponse juste; si plusieurs réponses sont valides, sélectionner la plus restrictive (par exemple s'il est demandé si 0 est *nul*, *non nul*, *positif*, ou *négatif*, sélectionner *nul*). Il n'est pas possible de corriger une erreur. Les réponses justes créditent; les incorrectes pénalisent; et les blanches et réponses multiples valent 0.

Nom et prénom :

.....

.....

.....

.....

Cochez votre identifiant (de haut en bas):

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

**Q.1** Un compilateur large

☐ ne peut pas détecter les fonctions inutilisées ☐ compile les langages des main-frames
☐ fait passer toutes les étapes au programme entier les unes après les autres
☐ fait passer toute la chaîne de traitement à chaque ligne de programme l'une après l'autre

**Q.2** Lex/Flex sont des

☐ scanners ☐ parseurs ☐ générateurs de scanners ☐ générateurs de parsers

**Q.3** Yacc/Bison sont des

☐ générateurs de parsers ☐ scanners ☐ parseurs ☐ générateurs de scanners

**Q.4** Comment désambiguiser pour Yacc/Bison le morceau d'arithmétique suivant :

```
exp: exp '+' exp | exp '-' exp | NUM;
```

☐ `%left '+' '-'`
☐ `%left '-' %left '+'`
☐ `%left '+' %left '-'`
☐ `%left '+' %left '-' %nonassoc NUM`

**Q.5** Comment désambiguiser pour Yacc/Bison le morceau d'arithmétique suivant :

```
exp: exp '*' exp | exp '+' exp | NUM;
```

☐ `%left '+' %left '*'`
☐ `%left '*' %left '+'`
☐ `%left '+' '*'`
☐ `%left '*' %left '+' %nonassoc NUM`

**Q.6** La syntaxe concrète est

☐ l'interface homme machine d'un langage de programmation
☐ une représentation des programmes à partir d'objets et de classes
☐ une grammaire de Backus en forme de Naur partagée
☐ une méthode de modélisation pragmatique

**Q.7** Le rôle d'un parser est de

☐ éliminer les récursions terminales
☐ segmenter un flux de caractères en un flux de tokens
☐ faire de l'analyse syntaxique
☐ s'assurer que les types sont bien utilisés

**Q.8** Désucrer signifie

☐ reconnaître et corriger les erreurs de programmation typiques
☐ retirer les commentaires, signes de ponctuation et retour à la ligne
☐ traduire certaines constructions syntaxiques en une forme plus primitive
☐ convertir une grammaire de SUGAR (SUGAR Unleashes Grammar Attribute Rules) à YACC (Yet Another Compiler Compiler)

**Q.9** Que signifie AST ?

☐ Arbre de syntaxe abstraite
☐ Arbre de synthèse abstraite
☐ Arbre abstrait de synthèse
☐ Arbre abstrait de syntaxe

**Q.10** La classe `symbol` qui gère les identifiants permet de

☐ de s'affranchir des caractères ASCII
☐ de faire de la correction orthographique
☐ les manipuler aussi efficacement que s'il s'agissait d'entiers
☐ de faire de la reprise sur erreur

**Q.11** Quel rôle ne jouent pas les langages intermédiaires ?

☐ factorisation de certaines optimisations
☐ résolution de la surcharge
☐ indépendance des parties frontales et terminales
☐ décomposition en plusieurs étapes de la traduction

**Q.12** Que n'inclue pas le graphe de flot de contrôle ?

☐ le passage à l'instruction suivante
☐ les branchements conditionnels
☐ les appels de fonctions
☐ les branchements inconditionnels

**Q.13** Quelles sont les variables vivantes au sortir de ce code ?

```
    a := 0
L1: b := a + 1
a := b * 2
```

☐ a et b
☐ a
☐ b
☐ on ne peut pas le savoir

**Q.14** Pour le code d'une fonction, les registres callee-save sont

☐ live-in et live-out
☐ on ne peut pas le savoir
☐ live-in
☐ live-out

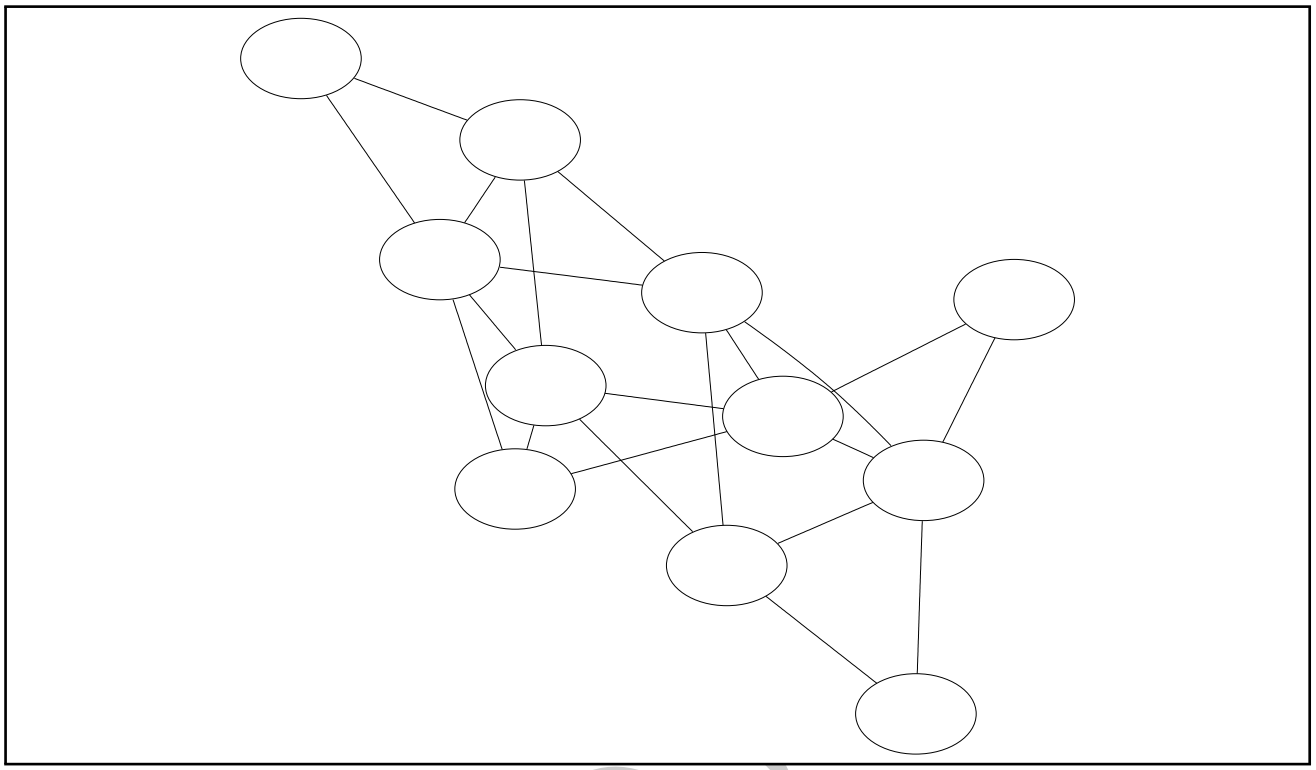
**Q.15** Que signifie le « spilling » ?

☐ la fusion deux temporaires
☐ la sauvegarde d'un registre callee-save
☐ l'allocation d'une temporaire dans un registre
☐ l'allocation d'une temporaire sur la pile

**Q.16** On peut colorer un graphe en quatre couleurs.

☐ toujours et même trois suffisent
☐ jamais
☐ toujours
☐ parfois

**Q.17** Colorer ce graphe d'exclusion mutuelle en 3 registres : R1, R2, R3. L'utilisation de couleurs différentes n'est pas requise, mais sera appréciée. ☐**0** ☐**1** ☐**2** ☐**3** ☐**4** *Reservé*



**Q.18** Que signifie le « coalescing » ?

☐ la fusion deux temporaires ☐ la sauvegarde d'un registre callee-save
☐ l'allocation d'une temporaire sur la pile ☐ l'allocation d'une temporaire dans un registre

On considère l'ajout en Tiger de l'opérateur '`??`' tel qu'on peut le rencontrer en C# : '`a ?? b`' s'évalue en `a` si `a` n'est pas « faux », et `b` sinon. Dans un premier temps, on ne s'intéressera qu'au cas où `a` et `b` sont des expressions entières.

**Q.19** Quelles règles ajouter au scanner Flex ?

☐
```
"??"    return TOK_QUESTION; return TOK_QUESTION;
```

☐
```
"??"    return TOK_QUESTION_QUESTION;
```

☐
```
"??"    yylval->text = strdup(yytext); return TOK_OPERATOR;
```

☐
```
%x SC_AFTER_QMARK
%%
"?"                    BEGIN SC_AFTER_QMARK;
<SC_AFTER_QMARK>"?"    return TOK_QUESTION_QUESTION;
```

**Q.20** Quelles règles ajouter au parser Bison ?

☐
```
exp ::= exp "?" "?" exp.
```

☐
```
exp: exp "??" exp %right;
```

☐
```
exp: exp "??" exp;
```

☐
```
exp: exp "??" exp %left;
```

☐
```
exp ::= exp "??" exp.
```

**Q.21** Combien existe-t-il d'arbres de dérivation pour '`a ?? b ?? c`' ?

☐ 3 ☐ 2 ☐ 4 ☐ 0 ☐ 1

**Q.22** Étant donnée sa sémantique, cet opérateur doit être rendu :

☐ associatif à gauche ☐ non-associatif ☐ associatif à droite
☐ idempotent ☐ commutatif

**Q.23** On le rendra associatif à droite, et moins prioritaire que les opérateurs logiques '|' et '&' (ce dernier étant prioritaire sur le précédent). Quelles directives Bison utiliser ?

☐
```
%right "??"
%left "|"
%left "&"
```

☐
```
%left "|"
%left "&"
%right "??"
```

☐
```
%left "&"
%left "|"
%right "??"
```

☐
```
%right "??"
%left "&"
%left "|"
```

☐
```
%right "??"
%left "|"
      "&"
```

**Q.24** Comment désucrer '`a ?? b`' si `a` et `b` sont des expressions quelconques de type `int` ?

☐
```
let
  var a := a
in
  if a <> 0 then a else b
end
```

☐
```
let
  var test  := a <> 0
  var true  := a
  var false := b
in
  if test then true else false
end
```

☐
```
if a then a else b
```

☐
```
let
  var a := a
  var b := b
in
  if a <> 0 then a else b
end
```

☐
```
let
  var b := b
  var a := a
in
  if a <> 0 then a else b
end
```

**Correction:** Il fallait remarquer que dans le cas où `a` et `b` sont des expressions de type entier, alors elles peuvent quand même être des fonctions avec des effets de bord, auquel cas la seul bonne réponse est la (e). Toutes les autres, ou bien peuvent évaluer `a` deux fois, ou bien évalue `b` même quand il ne le faut pas, voire évalue même `b` avant `a`.
Si cependant `a` et `b` sont des litéraux entiers, toutes les réponses sont justes. Par conséquent dans la notation, cette question a un coefficient nul.

**Q.25** Quelles sont les règles de typage de '`a ?? b`' ?

☐ `a` entier ou nul, et `b` entier non nul, retourne non nul
☐ `a` et `b` entiers, retourne entier
☐ `a` entier, retourne entier

**Q.26** Étant donnés ce désucrage et ces règles de typage (voir les deux questions précédentes), il n'est pas nécessaire de faire un contrôle de type spécifique pour '`??`'.

☐ vrai ☐ faux

**Q.27 ♣** En conséquence, quels modules du compilateur faut-il modifier pour supporter l'opérateur '`??`' sur les entiers ?

☐ scanner ☐ typage ☐ code intermédiaire ☐ parser ☐ liaison de noms

**Q.28** On désire à présent que cet opérateur fonctionne également avec les chaînes de caractères : '`a ?? b`' retourne `b` si `a` est la chaîne vide, `a` sinon. Quelles sont les règles de typage de '`??`' ?

☐ prend deux chaînes ou deux entiers, et renvoie une chaîne

☐ prend deux chaînes ou deux entiers, et renvoie une entier

☐ prend deux chaînes et renvoie une chaîne ou bien prend deux entiers et renvoie un entier

**Q.29** Comment désucrer '`a ?? b`' si `a` et `b` sont des expressions de type chaîne de caractères ?

☐
```
let
  var a := a
in
  if size(a) then a else b
end
```

☐
```
let
  var a := a
  var b := b
in
  if size(a) <> 0 then a else b
end
```

☐
```
let
  var b := b
  var a := a
in
  if size(a) <> 0 then a else b
end
```

☐
```
let
  var test  := size(a) <> 0
  var true  := a
  var false := b
in
  if test then true else false
end
```

☐
```
if size(a) <> 0 then a else b
```

**Correction:** À nouveau la rédaction originale pouvait laisser penser qu'il s'agissait de chaînes de caractères litérales, auquel cas toutes les réponses étaient juste. Question annulée dans la notation.

**Q.30 ♣** Quels sont les modules concernés si l'on désire introduire cet opérateur sur les entiers et les chaînes de caractères ?

☐ parser ☐ scanner ☐ liaison de noms ☐ typage

**Correction:** Il faut désormais connaître le type des arguments pour résoudre l'appel, comme c'est déjà le cas par exemple pour '<' ou encore =. Il faut donc retarder le désucrage à après le typage.

**Q.31** On désire également pouvoir appliquer cet opérateur sur les enregistrements (les *records* de Tiger). Quelles sont les règles de typage générales de '`a ?? b`' ?

☐ `a` et `b` doivent être du même type (entiers, ou chaînes, ou mêmes types d'enregistrements), et c'est le type du résultat.

☐ `a` et `b` doivent être du même type, et c'est le type du résultat ;

☐ *a* et *b* doivent être simultanément des entiers, des chaînes ou des enregistrements, et c'est le type du résultat ;

**Q.32** Il convient de ne pas confondre une expression qui *vaut* '`nil`' (par exemple une variable déclarée avec un type enregistrement spécifique, mais initialisée à '`nil`') d'une expression dont le type *est* Nil (comme '`nil`' elle-même). On souhaite supporter '`nil`' également. Comment traduire '`a ?? b`' quand *a* est du type Nil ?

☐
```
b
```

☐
```
(a; b)
```

☐
```
let
  var a := a
in
  if a <> nil then a else b
end
```

**Fin de l'épreuve.**