Université Mohammed V – Souissi
École Nationale Supérieure d'Informatique et d'Analyse des Systèmes

EXAMEN

Année Universitaire : 2012 - 2013
Filière : Ingénieur
Semestre : S3 - **Période :** P2

Module : **M3.4** - Compilation
Élément de Module : **M3.4.1** - Compilation
Professeur : Karim BAÏNA

- **Date : 09/01/2013**
- **Durée : 1H30**

Consignes aux élèves ingénieurs :
- *Seule la fiche de synthèse (A4 recto/verso) est autorisée !!*
- *Le barème est donné seulement à titre indicatif !!*
- *Soignez votre **présentation** et **écriture** !!*

Exercice I : Question de cours *(5pts)*

Concept/Question	Choix unique	Choix possibles
(I. **1**) Une grammaire LL(1) est forcément récursive droite	**F**	(**V**) Vrai (**F**) Faux
(I. **2**) Une grammaire récursive gauche peut coder une associativité droite	**V**	(**V**) Vrai (**F**) Faux
(I. **3**) Une grammaire attribuée récursive droite peut produire à un attribut AST gauche	**V**	(**V**) Vrai (**F**) Faux
(I. **4**) Une grammaire attribuée récursive droite pour produire un attribut AST gauche doit être S-attribuée	**V**	(**V**) Vrai (**F**) Faux
(I. **5**) Une grammaire arithmétique LL(1) L-attribuée est structurée pour produire un attribut AST gauche	**F**	(**V**) Vrai (**F**) Faux

Exercice II : Syntaxe, Sémantique et Représentations intermédiaires *(5 pts)*

1) Soit la nouvelle grammaire LL(1) des instructions ZZ :

INST : ... | for IDF = INUMBER to INUMBER do LISTE_INST endfor | switch '(' IDF ')' SWITCH_BODY endswitch	**SWITCH_BODY** : case INUMBER ':' LISTE_INST break ';' SWITCH_BODY | default ':' LISTE_INST break ';' **IF_INSTAUX** : endif | else LISTE_INST endif

Et soit le nouveau types Type_INST et INST pour stocker la représentation intermédiaire d'une instruction :
typedef enum {AssignArith, AssignBool, IfThenArith, IfThenElseArith, PrintIdf, For, Switch} Type_INST ;

```
typedef struct INST {
  Type_INST typeinst;
  union  { //...
    // for index = nb_min..nb_max do list_inst endfor
    struct {
      int rangvar; // indice de l'index de la boucle
      int borneinf; // l'expression borne inf
      int bornesup; // l'expression borne sup
      struct LIST_INST * forbodylinst; // for body list of instructions
    } fornode ;
    // switch ( x ) case 1 :  list_inst break ; case 20 : list_inst break ; …. default : list_inst break ; endswitch
    struct {
      int rangvar;                  // indice de la variable du switch
      int nbcases ;                 // taille du tableau dynamique cases suivant
      struct case *cases            // pour les cases (SWITCH_BODY), tableau dynamique non trié de couples (value, list_inst)
      struct LIST_INST * defaultbodylinst ; // la liste d'instructions par défaut du switch
    } switchnode ;
} node;
} instvalueType ;

typedef struct case {
            int value ;                                 // la valeur du cas (doit être >= 0)
            struct LIST_INST * casebodylinst; // la liste d'instructions du case
} casevaluelinst;
```

Compléter la fonction suivante d'interprétation de la représentation intermédiaire graphique (noeud du CFG) d'une instruction :

```
void interpreter_inst(instvalueType instattribute){
  double rexp;
  switch(instattribute.typeinst){
...
  case For :
  int i ;
  for (i = instattribute.node.fornode.borneinf ;        i < = instattribute.node.fornode.bornesup                    ; i++) {
            set_valinit(instattribute.node.fornode.rangvar, i) ;
            interpreter_list_inst( instattribute.node.fornode.forbodylinst ) ;                                  }  break;
  case Switch :
  int i = 0 ;
  while( (i < instattribute.node.switchnode.nbcases) &&
         (valinit(instattribute.node.switchnode.rangvar) != instattribute.node.switchnode.cases[i].value) i++ ;

  if (i < instattribute.node.switchnode.nbcases) { interpreter_list_inst( instattribute.node.switchnode.cases[i].casebodylinst ) ; }
  else { interpreter_list_inst( instattribute.node.switchnode.defaultbodylinst ) ; }  break;
 } // fin switch
}// fin interpreter_inst
```

Exercice III : Machine Virtuelle, Génération et Interprétation de pseudo-code *(10 pts)*

Code source ZZ	Pseudo-code 1 adresse généré pour un AST d'associativité gauche (**à compléter**)	Pseudo-code 1 adresse généré pour un AST d'associativité droite (**à compléter**)
demiRayon INT 4; Perimetre DOUBLE 0; Surface DOUBLE 0.0; Pi DOUBLE 3.14; **BEGIN** Perimetre = 2.0 * Pi * 2 * demiRayon; print Perimetre; Surface = ((Pi * (4 * demiRayon)) * demiRayon); print Surface; **END**	demiRayon 4.000000 Perimetre 0.000000 Surface 0.000000 Pi 3.140000 **begin:** PUSH 2.000000 **(1 pt)** LOAD Pi MULT PUSH 2.000000 MULT LOAD demiRayon MULT STORE Perimetre LOAD Perimetre PRINT LOAD Pi **(1 pt)** PUSH 4.000000 LOAD demiRayon MULT MULT LOAD demiRayon MULT STORE Surface LOAD Surface PRINT **end:**	demiRayon 4.000000 Perimetre 0.000000 Surface 0.000000 Pi 3.140000 **begin:** PUSH 2.000000 **(1 pt)** LOAD Pi PUSH 2.000000 LOAD demiRayon MULT MULT MULT STORE Perimetre LOAD Perimetre PRINT LOAD Pi **(1 pt)** PUSH 4.000000 LOAD demiRayon MULT MULT LOAD demiRayon MULT STORE Surface LOAD Surface PRINT **end:**

Code source ZZ d'un calcul de fibonacci	Pseudo-code 1 adresse généré (**à compléter**)

REM fibonacci
 	
REM grand pere Fibo(i=0) = 1
gp int 1;
 	
REM pere Fibo(i=1) = 1
p int 1;
 	
REM petit fils
pf int 0;
 	
i int;
 	
correct bool;
 	
begin
 	
REM calcul de Fibo(i=10)
for i = 2 to 10 do
 	
pf = p + gp;
 	
gp = p;
 	
p = pf;
 	
rem print pf;
 	
endfor
 	
print pf;
 	
if (pf == 89) then
  correct = true;
else
  correct = false;
endif
 	
print correct;
 	
end
	gp 1.000000
	p 1.000000
	pf 0.000000
	i 0.000000
	correct 0.000000 **(1 pt)**
	begin:
	PUSH 2.000000
	STORE i
	for0:
	PUSH 10.000000 **(1 pt)**
	LOAD i
	JG endfor0
	LOAD p
	LOAD gp
	ADD
	STORE pf
	LOAD p
	STORE gp
	LOAD pf
	STORE p
	PUSH 1.000000
	LOAD i
	ADD **(1 pt)**
	STORE i
	JMP for0
	endfor0:
	LOAD pf
	PRINT
	PUSH 89.000000 **(1 pt)**
	LOAD pf
	JNE else1
	PUSH 1.000000
	STORE correct
	JMP endif1
	else1: **(1 pt)**
	PUSH 0.000000
	STORE correct
	endif1: **(1 pt)**
	LOAD correct
	PRINT
	end: