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

Examen

Année Universitaire : 2011 - 2012
Filière : Ingénieur
Semestre : S3
Période : P2

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

- **Date : 12/01/2011**
- **Durée : 2H00**

Consignes aux élèves ingénieurs :

- *Le barème est donné seulement à titre indicatif !!*
- *Les **réponses directes** et **synthétiques** seront appréciées*
- *Soignez votre **présentation** et **écriture** !!*

Exercice I : Questions de cours *(10 pts)*

Concept/Question	Choix unique	Choix possibles
(**1**) Bytecode Java	(**B**)	**(A)** démontrer qu' « une grammaire est ambiguë » est décidable mais l'inverse est non décidable
(**2**) **ADDOP** REG1, REG2	**J**	(**B**) Représentation.....**Linéaire**
(**3**) Nombre de Registres nécessaires pour une expression arithmétique	**I**	(**C**) Analyseur Ascendant
(**4**) Grammaire LL	**F**	(**D**) Erreur Syntaxique
(**5**) **A**corn **R**ISC **M**achine-ARM	**U**	(**E**) **L**ook **A**head **L**eft-to-right with **R**ightmost parse
(**6**) Select * From * ;	**D**	(**F**) Analyseur Descendant
(**7**) LALR	(**E**)	(**G**) Erreur Sémantique
(**8**) *(null).suivant	**G**	(**H**) Erreur Lexicale
(**9**) Grammaire LR	**C**	(**I**) Attribut nécessaire à la génération de pseudo-code
(10) Génération de code à une adresse	**P**	(**J**) Two-address code
(**11**) Grammaire Ambiguë	**K**	(**K**) Analyse floue
(**12**) Récursivité Gauche	**N**	(**L**) Analyse descendante non optimale
(**13**) Grammaire Héréditairement-ambiguë	**Q**	(**M**) Analogie avec les epsilon-NFA
(**14**) Grammaire non LL	**L**	(**N**) Analyse sans fin
(**15**) Grammaire algébrique	**S**	(**O**) Union de deux langages hors-contexte
(**16**) Grammaire à Terminaux nullables	**M**	(**P**) Stack Machine code
(**17**) n'est pas hors-contexte	**O**	(**Q**) Analyse impossible
(**18**) $({}^{n}\{{}^{m})^{n}\}^{m}$	**R**	(**R**) Analyse hors contexte impossible
(**19**) Commentaire C avec /* sans */	**H**	(**S**) Equation linéaire
(**20**) Dérivation droite et gauche	**T**	(**T**) Tri et tri inverse des feuilles
(11) semi-décidabilité	**(A)** « ***résolue*** »	(**U**) Three-address code

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

1) Améliorer la grammaire LL(1) des instructions ZZ pour prendre en compte l'instruction switch entière :
switch (x) case 1 : … break ; case 20 : … break ; …. default : … break ; endswitch
On notera que le case n'est pas obligatoire, mais, le default est toujours obligatoire à la fin. On supposera que la partie lexicale est déjà réalisée pour les nouveaux terminaux nécessaires.

```
INST : IDF = ADDSUB ';'
     | IDF = TRUE ';'
     | IDF = FALSE ';'
     | if '(' idf == ADDSUB ')' then LISTE_INST IF_INSTAUX
     | print IDF ';'
     | 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 ';'
```

** une autre solution peut être de rendre SWITCH_BODY nullable et gérer le default dans la règle appelante.

```
IF_INSTAUX :  endif  | else LISTE_INST endif
```

2) Compléter le type INST pour stocker la représentation intermédiaire du switch :
Soit le nouveau type d'instruction :

```
typedef enum {AssignArith, AssignBool, IfThenArith, IfThenElseArith, PrintIdf, For, Switch} Type_INST ;

typedef struct INST {
  Type_INST typeinst;
  union  {
 //...
```

```
  // for (index:= exp_min..exp_max) loop list_inst end loop;
  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 :  ... break ; case 20 : ... break ; .... default : ... break ; endswitch
   struct {
     int rangvar;              // indice de la variable du switch
     struct case *cases        // pour les cases (SWITCH_BODY), tableau dynamique non trié de couples val- liste
     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 cas
} casevaluelinst;
```

* Une autre solution est de coder le default comme dernier élément de la liste cases avec une valeur (value) impossible (négative, ex. -1). Pour cette solution, le tableau **cases** va toujours contenir au moins un couple !
** la structure **case** peut également contenir un pointeur struct case * nextcase ; (si l'on ne veut pas, à chaque découverte d'un cas, effectuer des realloc de tout le tableau **cases**, mais seulement une allocation du **nextcase**.

Exercice III : Machine Virtuelle, Génération et Interprétation de pseudo-code *(5 pts)*

Nous souhaitons optimiser le temps de branchement de l'interpréteur du pseudo-code.

Soient les nouvelles structures de stockage des représentations intermédiaires linéaires du pseudo-code

```
// structure des nom-valeur DATA
struct namevalue {
          char * name;
          double value;
};

// nouvelle structure pour les branchements
struct jump {

          char * label_name;// nom du label
          pseudocodenode * jmpto;// @ de ce label
};

// structure linéaire du pseudocode
struct pseudocodenode{
  struct pseudoinstruction first;
  struct pseudocodenode * next;
};
```

```
// nouvelle structure pour les opérandes
typedef union {
  char * var;
  double _const;
  struct jump jp;
  struct namevalue nv;
} Param;


// structure pour les pseudoinstructions 1 adress
struct pseudoinstruction{
  CODOP codop;
  Param param;
};

typedef struct pseudocodenode * pseudocode;
```

1) Compléter la nouvelle fonction interpréteur d'un pseudocode

```
// precondition pc <> NULL
void interpreter_pseudo_code(pseudocode pc){
struct pseudocodenode** next_label_adress=(struct pseudocodenode**)malloc(sizeof(struct pseudocodenode *));

 if (pc != NULL){
    interpreter_pseudo_instruction(pc->first, next_label_address);
    if (*next_label_adress == NULL) interpreter_pseudo_code(pc->next); // Il n y a pas de branchement !!
    else interpreter_pseudo_code(*next_label_address); // effectuer un branchement en O(1) si // JNE, JG ou
JMP
    }}
```

2) Compléter la fonction interpréteur d'une pseudo instruction

```
void interpreter_pseudo_instruction(struct pseudoinstruction pi, struct pseudocodenode ** next_label_adress){
  Element op1, op2 ;
  *next_label_adress = NULL;

  switch(pi.codop){
     case JNE:       op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
                     if (op1 != op2)  (*next_label_adress) = pi.param.jp.jmpto ; break;

    case JG:         op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
                     if (op1 > op2) (*next_label_adress) = pi.param.jp.jmpto ; break;

    case JMP:   (*next_label_adress) = pi.param.jp.jmpto ; break ;
     // interprétation des autres pseudo-instructions (hors scope)
  }}
```