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

Examen

Année Universitaire : 2009 - 2010
Filière : Ingénieur
Semestre : S3
Période : P2

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

- **Date : 15/01/2010**
- **Durée : 2H00**

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

Exercice I : Syntaxe et Représentations intermédiaires *(20 pts)*

Soit la grammaire LALR du langage ZZ

PROG : LISTE_DECL LISTE_INST ;
LISTE_DECL : DECL | LISTE_DECL DECL;
DECL : **idf** TYPE CONST_IB ;
TYPE : **int** | **double** | **bool** ;
CONST_IB : **iconst** | **dconst** | TRUEFALSE ;
TRUEFALSE : **true** | **false** ;
LISTE_INST : INST | LISTE_INST INST ;
INST : **idf** "**:=**" EXPA /* *Affectation arithmétique*/ |
if '**(**' IDF '**=**' EXPA '**)** **then** LISTE_INST **endif** /* *Conditionnelle arithmétique*/ |
if '**(**' IDF '**=**' EXPA '**)** **then** LISTE_INST **else** LISTE_INST **endif** |
PRINT **idf** ; /* *Affichage d'une variable* */
EXPA : EXPA '**+**' EXPA | EXPA '**-**' EXPA | EXPA '*****' EXPA | EXPA '/' EXPA | '**(**' EXPA '**)**' | **iconst** | **dconst** | **idf** ;

Avec les priorités usuelles et associativités gauches des opérateurs arithmétiques '+', '-', '*' et '/'

1. Ajouter à la grammaire l'instruction d'affichage d'une chaîne de caractère (2pts)
Exemple, le programme : INT X 11 **PRINT** "# *X* = " **PRINT** X **PRINT** "#\n" produit : # X = 11 #

2. Ajouter à la grammaire l'instruction d'affectation booléenne complexe (2pts)
Exemple : x := (x and y or not z)
%left or
%left and
%left not

3. Après l'enrichissement de la question (2) (a) que remarquez – vous, (b) que proposez-vous ? (2pts)

4. Ajouter à la grammaire la conditionnelle booléenne (2pts)
Exemple : if (x = true)… if (x = false) … if (x = ((not x) and (y or z)))

5. Après l'enrichissement de la question (4) (a) que remarquez – vous, (b) que proposez-vous ? (2pts)

6. Enrichir les types suivants pour prendre en compte les enrichissements I.1, I.2 et I.4 (2pts)
On supposera défini ASTB (par analogie à ASTA type des arbres abstraits arithmétiques) le type des arbres abstraits booléens.

```
typedef struct INST {
  Type_INST typeinst;
  union  {
    // idf := EXPA
    struct  {
      int rangvar; // indice de l'idf (left exp), où il faut affecter, dans la table des symboles
      ASTA right; // l'expression arithmétique droite (right exp) à affecter
    } arithassignnode;
    // if ... then  … else
    struct  {
      int rangvar; // indice de l'idf (left exp) à comparer, dans la table des symboles
      ASTA right; // l'expression arithmétique (right exp) à comparer
      struct LIST_INST * thenlinst; // then list of instructions
      struct LIST_INST * elselinst; // else list of instructions
    } ifnode;
    // PRINT idf
    struct  {
      int rangvar; // indice de l'idf (à afficher) dans la table des symboles
    } printnode;
  } node;
} instvalueType;
```

```
typedef struct LIST_INST {
  struct INST first;
  struct LIST_INST * next;
} listinstvalueType;

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

7. Donner 4 erreurs sémantiques différentes engendrées par les enrichissements I.2 et I.4 (2pts)

8. Nous voudrions pouvoir exprimer des comparaisons riches et les utiliser dans les affectations et les conditionnelles
BOOL x FALSE
Exemples d'affectations booléennes : x := (l <= (50+ y * y)) ou x := ((25 * m) >= (50+ y * y)) ou x := (z = true) ou x := ((z or f) = true)
Exemples de conditionnelles : if (x) … ou if (l <= (50+ y * y)) ou if ((z or f) = true)
Les opérateurs de comparaisons supportés (=, <=, >=).
Modifier la grammaire pour prendre en compte cet enrichissement (2pts)

9. Enrichir les types de la question I.6 pour prendre en compte les enrichissements I.8 (2pts)

10. Les représentations intermédiaires graphiques produites à la fin de la phase d'analyse sont-elles vraiment indispensables puisque nous pouvons nous en passer pour générer le pseudo-code en même temps que l'analyse syntaxico-sémantique sans utiliser ni AST, ni DAG, ni CFG, … (*syntax driven translation*) (2pts)

Exercice II : Machine Virtuelle et Génération de pseudo-code *(10 pts, dont au max 4 de bonus TP)*

Soit l'instruction for dont la syntaxe est la suivante :
INST : **for idf ":="** nombre **to** nombre **loop** LIST_INST **end loop ;** | …

Son type d'instruction : typedef enum {…. forLoop } Type_INST ;

Et sa représentation intermédiaire (faisant part du type *node*)

```
typedef struct INST {
        Type_INST typeinst;
        union { …. // les autres types d'instructions
                // for idf ":=" nombre to nombre loop LIST_INST end loop ;
                struct  {
                int rangvar; // indice de l'idf (variable d'induction de la boucle à comparer) dans la table des symboles
                int min; // la valeur de la borne inférieure de l'intervalle d'itération
                int max; // la valeur de la borne supérieure de l'intervalle d'itération
                struct LIST_INST * forbodylinst; // la liste d'instructions corps de la boucle pour
                } fornode;
        } node;
} instvalueType;
```

Nous rappelons les structures de base :

```
typedef enum {ADD, DIV, DUPL, JMP, JNE, JG, LABEL, LOAD, MULT, POP, PRNT, PUSH, SUB, STORE, SWAP} CODOP;

typedef union {
  char * var;            // pour LOAD / STORE
  double _const;         // pour PUSH
  char * label_name; // pour JMP/JNE/JG/LABEL
} Param ;

typedef struct pseudocodenode * pseudocode;

struct pseudoinstruction{
        CODOP codop ;
        Param param ; // une opération possède un paramètre au maximum
} ;
struct pseudocodenode{
        struct pseudoinstruction first ;
        struct pseudocodenode * next ;
} ;
```

Comme vu en cours, la fonction **void interpreter_list_inst(listinstvalueType * plistinstattribute)** et
La fonction **pseudocode generer_pseudo_code_list_inst(listinstvalueType * plistinstattribute)** sont déjà définies.

1. Compléter l'interpréteur de représentations intermédiaire pour prendre en compte l'instruction for : (2pts)

```
void interpreter_inst(instvalueType instattribute){
        switch(instattribute.typeinst){
                case forLoop :
                        // à compléter ….
                break ;
        } // end switch

}
```

2. Compléter le générateur de code pour prendre en compte l'instruction for : (2pts)

```
pseudocode generer_pseudo_code_inst(instvalueType instattribute){
pseudocode  pc = (pseudocode)malloc(sizeof (struct pseudocodenode));
        switch(instattribute.typeinst){
                case forLoop :
                        // à compléter ….
                break ;
                } // end switch
        return pc ;
}
```

3. (i) Spécifier les profils des fonctions du type abstrait de données pile (empiler, dépiler, tête_pile, taille_pile, pile_vide) **sans les implémenter. (ii) Supposer l'existence d'une pile système globale** pile VM_STACK ; **liée à la machine virtuelle, (iii) Réaliser l'interpréteur du pseudo-code intégré à la machine virtuelle à travers les deux fonctions :** (2pts)

```
void interpreter_pseudo_code_inst(pseudoinstruction pc) ;
void interpreter_pseudo_code_list_inst(pseudocode pc) ;
```