Université de Lille — Faculté des Sciences et Technologies — Département Informatique

Informatique

L3S5 Miage

Analyse Syntaxique – TP5

Initiation à la compilation

Ce TP a pour objectif de créer un compilateur MICROCOBOL vers JAVA.

Le MICROCOBOL est un dérivé du COBOL (*COmmon Business Oriented Language*). Comme son nom le suggère subtilement, le langage COBOL est destiné à la programmation d'applications de gestion.

La création du compilateur MICROCOBOL durera sur plusieurs séances de TPs. Conservez votre code !

1. microcobol : création des classes

Dans cette partie, vous allez construire *à la main* un arbre de dérivation d'un programme en MICROCOBOL. Commençons par présenter le langage.

1.1. Structure et variables

Un programme MICROCOBOL est constitué de 3 divisions, elles-mêmes constituées de sections. Les trois divisions sont les suivantes :

- `IDENTIFICATION DIVISION`, obligatoire, qui contient le nom du programme ; elle contient le mot clé `PROGRAM-ID` suivie du nom du programme ;
- `DATA DIVISION`, facultative, contient les variables utilisées dans le programme ;
- `PROCEDURE DIVISION` contient les procédures du programme, appelées *paragraphes*.

Le programme se termine par `END PROGRAM` suivi du nom du programme.

Les paragraphes commencent par leur nom suivi d'un point, puis d'une liste de *phrases* (les instructions) se terminant chacune par un points.

Voici un exemple de programme MICROCOBOL.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID programme-test.
PROCEDURE DIVISION.
  hello-world.
    DISPLAY "Hello World !".
    STOP RUN.
END PROGRAM programme-test.
```

Ce programme s'appelle `programme-test` (oui, vous avez bien lu, il y a un tiret `-` dans le nom du programme !), ne déclare aucune variable (pas de division `DATA DIVISION`), et définit un unique paragraphe nommé `hello-world`. Ce paragraphe contient deux phrases. La première affiche le message « Hello World ! », la seconde met fin à l'exécution du programme.

La division `DATA DIVISION` facultative contient la *section* `WORKING-STORAGE SECTION`. Cette section contient la liste des variables utilisées dans le programme sous la forme `N nom PIC forme` où :

- `N` est 77,
- `nom` est le nom de la variable,
- `PIC` est un mot clé,
- `forme` est le *type* de la variable donné sous une forme dont nous verrons la signification plus tard.

Dans l'exemple suivant, on définit deux variables, qui ne sont pas utilisées.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID an2000
DATA DIVISION.
  WORKING-STORAGE SECTION.
    77 date PIC 99/99/99.
    77 nom  PIC AAAAAAAAAA.
PROCEDURE DIVISION.
  main.
    STOP RUN.
END PROGRAM an2000.
```

La variable `date` contiendra 6 chiffres, alors que la variable `nom` contiendra 10 lettres.

1.2. Phrases et instructions

Les instructions en MICROCOBOL commencent le plus souvent par un *verbe*, sinon par un mot-clé. Une phrase est une instruction qui finit par un point.

1.2.1. Affectation

Le verbe `MOVE` permet d'affecter une valeur littérale ou la valeur d'une variable à une autre variable.

```
MOVE x TO y.
```

Cette phrase va réaffecter à la variable `y` la valeur de `x`.

1.2.2. Arithmétique

Les verbes `ADD`, `SUBSTRACT`, `MULTIPLY` et `DIVIDE` permettent respectivement d'additionner, de soustraire, de multiplier et de diviser.

```
ADD 1 TO x.
SUBSTRACT y FROM x.
MULTIPLY y BY x.
DIVIDE y INTO x.
```

L'exemple précédent est équivalent au code JAVA suivant.

```
x += 1;
x -= y;
x *= y;
x /= y;
```

1.2.3. Conditions

Les conditions s'écrivent sous la forme `x IS [NOT] op y` où :
— `x` et `y` sont des identificateurs de variables ou des littéraux,
— `NOT` est un mot clé facultatif,
— `op` est un opérateur de comparaison (à choisir parmi `=`, `<` ou `>`).
Attention, une condition n'est pas une phrase (elle ne commence pas par un verbe et ne finit pas par un point) !

1.2.4. Branchement conditionnel

Une phrase de branchement s'écrit sous la forme :

```
IF condition
  THEN instructions
  [ELSE instructions]
END-IF.
```

— `condition` est une condition,
— le `ELSE` suivi des instructions est facultatif,
— les `instructions` sont des instructions.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. ex-condi.
PROCEDURE DIVISION.
  main.
    IF 1 IS < 2
      THEN DISPLAY "1 < 2"
      ELSE DISPLAY "1 >= 2"
    END-IF.
    STOP RUN.
END PROGRAM ex-condi.
```

1.2.5. Boucle

```
PERFORM UNTIL condition
  instructions
END-PERFORM.
```

— `condition` est une condition,
— les `instructions` sont des instructions.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. compte-a-rebours.
DATA DIVISION.
  WORKING-STORAGE SECTION.
    77 compteur 99.
PROCEDURE DIVISION.
  main.
    MOVE 10 TO compteur.
```

```
    PERFORM UNTIL compteur IS = 0
      DISPLAY compteur
      SUBSTRACT 1 FROM compteur
    END-PERFORM.
    DISPLAY "Boom !"
    STOP RUN.
END PROGRAM compte-a-rebours.
```

1.2.6. Affichage

Le verbe `DISPLAY` affiche la valeur des variables ou des littéraux qui le suivent.

```
DISPLAY "Bonjour " nom-du-sujet " !".
DISPLAY "Vous avez mis moins de " n " minutes pour résoudre le test."/
DISPLAY "Vous devez faire la fierté de " ville-du-sujet ".".
```

1.3. Création d'un arbre de dérivation

Vous trouverez dans le dépôt les fichiers nécessaires pour ce TP. Vous pouvez compiler les sources en vous placant dans le répertoire `src` et en lancant la commande `make`.

```
src$ make
java -jar /usr/share/java/jflex.jar microcobol.lex
Reading "microcobol.lex"
Constructing NFA : 437 states in NFA
Converting NFA to DFA :
..........................................................................................
224 states before minimization, 191 states in minimized DFA
Writing code to "Yylex.java"
java -jar /usr/share/java/cup.jar microcobol.cup
Warning : Terminal "FROM" was declared but never used
Warning : Terminal "MULTIPLY" was declared but never used
Warning : Terminal "SUBSTRACT" was declared but never used
Warning : Terminal "VALUE" was declared but never used
Warning : Terminal "GIVING" was declared but never used
Warning : Terminal "BY" was declared but never used
Warning : Terminal "DIVIDE" was declared but never used
Warning : Terminal "DISPLAY" was declared but never used
Warning : *** Production "listeinst ::= instruction " never reduced
Warning : *** Production "listeinst ::= listeinst instruction " never reduced
Warning : *** Production "listeobjets ::= objet " never reduced
Warning : *** Production "listeobjets ::= listeobjets objet " never reduced
------- CUP v0.11a beta 20060608 Parser Generation Summary -------
  0 errors and 12 warnings
  30 terminals, 19 non-terminals, and 32 productions declared,
  producing 74 unique parse states.
  8 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.java", and "sym.java".
----------------------------------------------------- (v0.11a beta 20060608)
mkdir -p classes/microcobol
```

```
javac -d classes -cp /usr/share/java/jflex.jar:/usr/share/java/cup.jar Objet.java Variable.java Li
Note: parser.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
jar -cvf microcobol.jar -C classes microcobol
manifeste ajouté
ajout : microcobol/(entrée = 0) (sortie = 0)(stockage : 0 %)
ajout : microcobol/Paragraphe.class(entrée = 1398) (sortie = 706)(compression : 49 %)
ajout : microcobol/DataDiv.class(entrée = 886) (sortie = 488)(compression : 44 %)
ajout : microcobol/Yylex.class(entrée = 10352) (sortie = 5766)(compression : 44 %)
ajout : microcobol/Condition.class(entrée = 1658) (sortie = 898)(compression : 45 %)
ajout : microcobol/MultiplyInst.class(entrée = 1016) (sortie = 541)(compression : 46 %)
ajout : microcobol/CUP$parser$actions.class(entrée = 7617) (sortie = 2940)(compression : 61 %)
ajout : microcobol/WorkingSect.class(entrée = 1258) (sortie = 684)(compression : 45 %)
ajout : microcobol/ProcDiv.class(entrée = 1275) (sortie = 684)(compression : 46 %)
ajout : microcobol/Chaine.class(entrée = 920) (sortie = 501)(compression : 45 %)
ajout : microcobol/DisplayInst.class(entrée = 1252) (sortie = 671)(compression : 46 %)
ajout : microcobol/sym.class(entrée = 1095) (sortie = 559)(compression : 48 %)
ajout : microcobol/Objet.class(entrée = 237) (sortie = 188)(compression : 20 %)
ajout : microcobol/Instruction.class(entrée = 249) (sortie = 199)(compression : 20 %)
ajout : microcobol/Variable.class(entrée = 360) (sortie = 248)(compression : 31 %)
ajout : microcobol/parser.class(entrée = 2876) (sortie = 1258)(compression : 56 %)
ajout : microcobol/IfInst.class(entrée = 1671) (sortie = 814)(compression : 51 %)
ajout : microcobol/Nombre.class(entrée = 362) (sortie = 251)(compression : 30 %)
ajout : microcobol/Data.class(entrée = 890) (sortie = 479)(compression : 46 %)
ajout : microcobol/IdDiv.class(entrée = 904) (sortie = 503)(compression : 44 %)
ajout : microcobol/Litteral.class(entrée = 201) (sortie = 162)(compression : 19 %)
ajout : microcobol/Phrase.class(entrée = 871) (sortie = 473)(compression : 45 %)
ajout : microcobol/MoveInst.class(entrée = 1004) (sortie = 537)(compression : 46 %)
ajout : microcobol/DivideInst.class(entrée = 1012) (sortie = 541)(compression : 46 %)
ajout : microcobol/LexicalException.class(entrée = 298) (sortie = 213)(compression : 28 %)
ajout : microcobol/Programme.class(entrée = 1322) (sortie = 671)(compression : 49 %)
ajout : microcobol/SubstractInst.class(entrée = 1021) (sortie = 546)(compression : 46 %)
ajout : microcobol/AddInst.class(entrée = 1001) (sortie = 535)(compression : 46 %)
ajout : microcobol/StopInst.class(entrée = 300) (sortie = 229)(compression : 23 %)
ajout : microcobol/OpComp.class(entrée = 1108) (sortie = 588)(compression : 46 %)
javac -cp /usr/share/java/jflex.jar:/usr/share/java/cup.jar:microcobol.jar TestParser.java
```

Il est possible de tester le parser dans l'état actuel en in voquant la commande :

```
src/$ java -jar /usr/share/java/cup.jar:microcobol.jar:. TestParser test1.cb
IDENTIFICATION DIVISION.
  PROGRAM-ID test1.
END PROGRAMME test1.
```

ou en utilisant le script `testparser.sh` fourni.

Question 1. *Complétez le code dans `microcobol.cup` pour ajouter les instructions arithmétiques `SUBSTRACT`, `MULTIPLY` et `DIVIDE`. Tester votre code sur le fichier `test2.cb`.*

Question 2. *Complétez le code dans `microcobol.cup` pour ajouter l'instruction conditionnelle `IF`. Tester votre code sur `test3.cb`*

Question 3. *Complétez le code dans `microcobol.cup` et `microcobol.lex` et rajoutez une classe `PerformInst.java` pour ajouter l'instruction de boucle `PERFORM`. Tester votre code sur `test4.cb`*

1.4. InstructionCOMPUTE (Facultatif)

Nous souhaitons compléter ke kangage en rajoutant une instruction `COMPUTE` permettant de calculer des expressions arithmétiques facilement.

```
COMPUTE x = (3 * y + 2) ^ 3 _ k / 12
```

Question 4. *Complétez le code en implémentant ce qu'il faut pour utiliser cette instruction. Tester votre code sur* `test5.cb`*. Vous pouvez créer un classe* `Expression` *et réutilisez le TP précédent.*