Projet 1 du cours INFO 010

THÉORIE DES LANGAGES ET DE LA COMPILATION

EMBEDDED C

Gilles GEERAERTS*

Année Académique 2007–2008

1 Introduction

Pendant de nombreuses années, le langage utilisé dans le développement des systèmes embarqués à coûts réduits était invariablement l'assembleur. Avec le temps et l'évolution technologique, ces microprocesseurs et micro-contrôleurs sont devenus de plus en plus sophistiqués. L'intérêt d'un langage de plus haut niveau s'est alors fait ressentir. Le langage C, que nous connaissons tous, étant très utilisé en programmation système, de nombreux constructeurs de systèmes embarqués ont tout naturellement décidé de le proposer comme outil de développement pour leurs processeurs, permettant ainsi aux développeurs de s'adapter très rapidement à ces nouveaux outils.

Mais le monde du micro-contrôleur n'est pas celui de la micro-informatique, et le langage C de donc été appauvri, prenant en compte les capacités limitées des systèmes embarqués, ainsi que le manque fréquent d'une gestion de mémoire digne de ce nom. C'est ainsi qu'est né l'EMBEDDED C...

2 L'EMBEDDED C

Plutôt qu'*un langage*, L'EMBEDDED C est plutôt une *famille de langages*, chaque langage de la famille étant adapté aux capacités d'un type de micro-contrôleur. La seule contrainte est que tout programme écrit en EMBEDDED C doit impérativement être un programme C syntaxiquement correct.

Le plus souvent, des pseudo-fonctions sont ajoutées au langage pour permettre d'interagir avec le monde réel et pour contrôler des fonctionnalités particulières du matériel : accès aux ports d'entrées-sorties du micro-contrôleur, instructions spécifiques (par exemple `reset()`, `sleep()`, `shutdown()`...) Une fois encore, ceci n'est pas fixé mais dépend de la plate-forme considérée.

Dans le cadre de ce projet, nous vous proposons de développer un compilateur pour l'EMBEDDED C dont la grammaire est donnée par la FIG. 1. Les non-terminaux ont pour formalisme <NONTERMINAL>, ID représente une variable C et CONST est un littéral représentant une constante numérique entière sans signe.

Des commentaires peuvent être insérés dans un programme en EMBEDDED C. Ils apparaîtront dans un style /* typiquement C */[1].

La figure 2 donne un exemple de programme en EMBEDDED C.

*Sur une idée originale de Sébastien COLETTE

[1]Remarquez qu'aucune règle n'est prévue pour les lignes de commentaires dans la grammaire.

[1] <PROGRAM> → <VARDECL> <MAIN>$
[2] <VARDECL> → <VAR> <VARDECL>
[3] → ε
[4] <VAR> → int ID ;
[5] <MAIN> → void main() <BLOCK>
[6] <BLOCK> → { <INSTRUCTIONLIST> }
[7] <INSTRUCTIONLIST> → <INSTRUCTION> <INSTRUCTIONLIST>
[8] → ε
[9] <INSTRUCTION> → <BLOCK>
[10] → <ASSIGNATION>
[11] → <IF>
[12] → <WHILE>
[13] → <READ>
[14] → <WRITE>
[15] <ASSIGNATION> → ID=<EXPRESSION> ;
[16] <EXPRESSION> → <EXPRESSION> <OP> <EXPRESSION>
[17] → (<EXPRESSION>)
[18] → -<EXPRESSION>
[19] → !<EXPRESSION>
[20] → ID
[21] → CONST
[22] <OP> → +
[23] → -
[24] → *
[25] → /
[26] → &&
[27] → ||
[28] → ==
[29] → <
[30] → >
[31] → <=
[32] → >=
[33] → !=
[34] <IF> → if(<EXPRESSION>)<BLOCK>
[35] → if(<EXPRESSION>)<BLOCK> else <BLOCK>
[36] <WHILE> → while(<EXPRESSION>)<INSTRUCTION>
[37] <READ> → read(ID) ;
[38] <WRITE> → write(<EXPRESSION>) ;

FIG. 1 – La grammaire d'EMBEDDED C

```
int a;
int b;
int c;

/* Tentative d'algorithme d'Euclide */

void main()
{
  read(a);
  read(b);

  while(b)
  {
    c=b;

    while(a >= b) /* Calcul de a modulo b */
      a=a-b;

    b=a;
    a=c;
  }
  write(a);
}
```

FIG. 2 – Un exemple de programme en EMBEDDED C

Il n'y a pas de piège : l'effet des instructions présentées ici est identique à celui observé en C, et la syntaxe est celle que nous connaissons bien.

Par contre, des éléments importants du C sont indisponibles : les fonctions, les déclarations de variables non-globales, le *for*,... De plus, l'unique type de données disponible est *int*, correspondant à un registre du micro-contrôleur.

Il faut donc considérer les opérations booléennes comme des opérations sur des nombres[2], 0 désignant arbitrairement le booléen *false* et les autres valeurs le booléen *true*. Donc, on peut écrire des expressions comme `!3`, dont la valeur est `0` ; ou bien `!0`, dont la valeur est 1.

Vous remarquerez également la présence de deux pseudo-fonctions, *write()* et *read()*, qui permettent respectivement d'afficher un nombre sur l'écran LCD relié au micro-contrôleur ou de lire un nombre à partir du clavier numérique relié à celui-ci.

3 Machine virtuelle

Le but d'un compilateur étant tout de même la génération de code, il est temps d'introduire la «machine» vers le langage de laquelle nous allons compiler les programmes. Il s'agit de la GPMACHINE, qui fonctionne avec une pile et qui interprète des instructions en PCODE. Cette machine et son jeu d'instructions ont été étudiés au TP.

Vous trouverez cette machine virtuelle à l'adresse : `http://www.info.fundp.ac.be/~gpm`. Un mode d'emploi pour GPMACHINE est disponible sur la page web du TP.

4 *Desiderata*

Étant donnée la grammaire de l'EMBEDDED C, nous vous demandons d'implémenter un compilateur pour ce langage. Vous pouvez choisir de l'implémenter en C ou en C++[3]. Plus précisément nous attendons de votre part le travail suivant :

1. Identifiez les *unités lexicales* de l'EMBEDDED C ;
2. Donnez un automate fini déterministe les reconnaissant ;
3. Écrivez un analyseur lexical, qui reconnaît les différentes unités lexicales de l'EMBEDDED C. Celui-ci sera implémenté sous forme d'une fonction qui, à chaque appel, retournera la prochaine unité lexicale lue sur l'*input*. Une table des symboles (qui contient tant les variables que les constantes littérales utilisées) sera également implémentée ;
4. Modifiez la grammaire de l'EMBEDDED C de telle sorte que :
   (a) Celle-ci soit LL(1) ;
   (b) Elle tienne compte de la *priorité* et de l'*associativité* des opérateurs, à savoir (du plus au moins prioritaire) :

[2] Ce qui ressemble étrangement au comportement du C !

[3] Si vous désirez l'implémenter dans un autre lagnage, consultez d'abord l'assistant.

Opérateur(s)	Associativité
( )	
!, −	à droite
⋆, /	à gauche
+, −	idem
>, <, >=, <=, !=, ==	idem
&&	idem
||	idem

5. Donnez la table de *parsing* LL(1) pour cette grammaire, ainsi que les éventuels calculs de First() et de Follow() qui vous auront été nécessaires ;
6. Décorez la grammaire en lui ajoutant les instructions nécessaires pour produire du code pour la machine virtuelle ;
7. Écrivez un analyseur LL(1) en descente récursive, produisant du code pour la machine virtuelle dans un fichier texte.

Le tout sera naturellement présenté dans un *rapport*, qui reprend les différentes informations demandées, et qui précise et justifie les choix et hypothèses que vous aurez faits. Le code source de votre compilateur (accompagné de ses *Makefile*, de ses fichiers de test et de tout autre fichier nécessaire pour se convaincre que le travail a été bien fait) doit être envoyé par e-mail à `gigeerae@ulb.ac.be`. Le tout doit être remis pour le vendredi 21 décembre. Le travail peut être fait par groupe de deux (ce que nous recommandons), ou individuellement.

Bon travail !