KarelParser

Marc-Antoine Lacasse

30 avril 2009

Résumé

karelparser est une librairie écrite en C++ servant à vérifier la syntax d'un program écrit en karel, le langage utilisé pour programmer les robots Fanuc. La librairie utilise boost : :spirit afin de générer les règles et la grammaire correspondant à la syntax du karel. Un itérateur de type position : :iterator permet de situer les erreurs lorsqu'une exception est lancée.

Table des matières

1 Introduction

J'ai débuté cette librairie lorsque j'avais comme projet de programmer un simulateur d'erreur de position pour robot sériel. Ce simulateur m'aurait permis de développer des algorithmes de compensation d'erreurs pour mon projet de maîtrise. Comme, à la base, le projet devait s'appliquer sur des robots Fanuc, j'ai souhaité que mon simulateur puisse interpréter les même fichiers de code karel que ceux avec lesquels je programmais mon robot. Je suis rapidement tombé sur la librairie Spirit qui semblait l'outils idéal pour réaliser un analyseur syntaxique.

2 Grammaires

Les grammaires sont un ensemble de règles définissant un élément du langage.

2.1 karel language grammar

Cette grammaire comprend les différentes règle de la syntax d'une fichier karel.

2.2 skip grammar

Cette grammaire permet de ne pas tenir compte des commentaires et des blancs lors de la lecture d'un fichier karel. En Karel, certain marqueur de fin de ligne sont obligatoires, d'autres optionnels, par exemple :

```
PROGRAM TEST
BEGIN
END TEST
```

est un fichier Karel minimal mais syntaxiquement correcte. Par contre, le code :

```
PROGRAM TEST BEGIN
END TEST
```

n'est pas acceptable puisque l'identifiant du program doit être suivie d'un marqueur de fin de ligne, ';' ou '\n'.

3 Exceptions

Un premier type d'exception est lancée lorsqu'une règle de syntax échoue. L'exeption lancée est simplement un entier correspondant au numéro de l'erreur. L'erreur n'est pas nécessairement fatale. Prenons par exemple la règles suivante : Si var_declaration échoue, une exception sera lancé contenant la description de l'erreur ainsi que son emplacement. L'exeption sera rattrappée puis l'erreur empilée avant que la règle const_declaration ne soit testé. const_declaration lancera à son tour une exception si elle échoue et l'erreur sera empilée. Par contre, s'il y a un match, la règle declaration matchera et la pile d'erreur sera vidé via l'action semantic clear_a().

```
declaration =
  guard<int> my_guard;
  my_guard
  (
    var_declaration
    |
    const_declaration
  )
  [clear_a(error_container_type::instance())];
```

4 Utilisation

4.1 utilisation générale

```
#include "karel.hpp"
#include <fstream>

using namespace std;
using namespace malac::karel;
using namespace BOOST_SPIRIT_CLASSIC_NS;

void main()
{

  typedef char char_t;
  typedef file_iterator <char_t>  iterator_t;

  iterator_t first( "fichier.kl" );
  iterator_t last = first.make_end();

  typedef position_iterator<iterator_t> pos_file_iterator_t;
  pos_file_iterator_t p_begin(first, last, filename);
  pos_file_iterator_t p_end;

  interpreter interpret;
  skip_grammar skip;
```

```
  tree_parse_info<pos_file_iterator_t> info = ast_parse(p_begin, p_end, interpret, skip);

  if(!info.full)
    std::cout << "Erreur : parsage incomplet" << std::endl;
  else
    std::cout << "Aucune erreur détecté" << std::endl;

};
```

4.2 programme Qt

J'ai débuté un programme minimaliste en Qt afin de rendre plus accessible la librairie karelParser. Le programme comprend une fenêtre principale avec un menu pour ouvrir et enregistrer un fichier ainsi que pour quitter le programme.

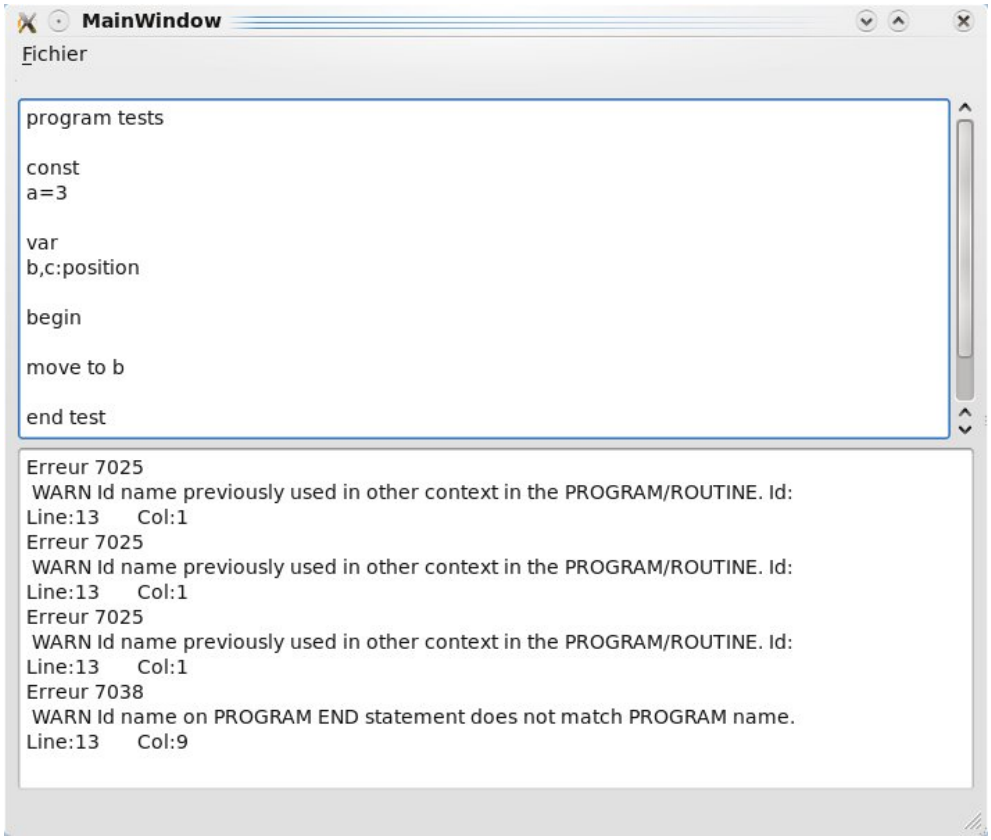

FIG. 1 – Fenêtre principale du programme

5 À faire

plusieurs choses...

6 Contact

Marc-Antoine Lacasse
malacasse@gmail.com