Resolução do Trabalho I

Linguagens de Programação

Gabriel Weich e Paulo Aranha

Escola Politécnica – PUCRS

Agosto de 2018

Introdução

A proposta deste estudo é apresentar uma gramática gerada pelo *framework* Xtext para reconhecer uma versão simplificada do comando *struct* da linguagem C++. Serão apresentados alguns exemplos de variações do comando aceitos pela gramática juntamente com alguns casos de teste para validação da gramática.

Comando Struct

O comando *struct* define uma coleção de variáveis de diferentes tipos agrupadas sobre um nome. É utilizado para representar o modelo de um objeto possível de ser instanciado.

A sintaxe do comando aceita pela nossa gramática pode ser definida tal como segue:

```
struct [identificador] : [base] {
   tipo [membro];
   tipo [membro];
   ...
} [declaradores];
```

O comando pode ser utilizado através da palavra reservada *struct* seguida por um nome identificador opcional, uma classe ou estrutura opcional da qual irá derivar seus membros, uma sequência de membros especificados por um tipo e um identificador dentro de um bloco de dados e uma lista opcional de declaração de objetos do tipo da estrutura.

Exemplos

Após a definição do comando, escolhemos alguns exemplos que serviriam de base para a criação da gramática.

1. O primeiro exemplo demontra uma declaração básica de um *struct*, incluindo a palavra chave, um identificador e dois membros. [1]

   ```
   struct X {
       int a;
       int b;
   };
   ```

2. O segundo exemplo demontra a possibilidade de declarar instancias de objetos do tipo da estrutura.[2]

```
struct product {
    int weight;
    double price;
} apple, banana, melon;
```

3. O terceiro exemplo demontra que é possível utilizar *arrays* tanto na declaração dos membros como na declaração dos objetos. Além disso, demontra a possibilidade de um membro ter como tipo outro *struct*. [2]

```
struct movies_t {
    string title;
    int year;
} films [3];

struct friends_t {
    char name[25];
    string email;
    movies_t favorite_movie;
} charlie, maria;
```

4. O quarto exemplo demonstra a possibilidade de um *struct* ter como base outro *struct* do qual herdará seus atributos. [3]

```
struct A { };
struct B : A { };
struct C : B { };
```

Gramática

Uma gramática em Xtext capaz de reconhecer os exemplos acima pode ser definida como segue:

```
Model:
    structs+=Struct*;


Struct:
    'struct' (name=ID)? (':' superType=[Struct])? '{'
        (members+=Member)* '}' (objects+=Object)? ";";


Member:
    (type=Type | stype=[Struct]) name=ID (array=Array)?";"
;

Array:
    "[" name=INT "]"
;
```

```
Object:
    name=ID (array=Array)? (',' objects=Object)*
;

Type:
    'char' | 'int' | 'bool' | 'float' | 'double' |
    'void' | 'wchar_t' | 'string'
;
```

Testes Automatizados

Durante e após a construção da gramática definimos alguns casos de teste para garantir o funcionamento do que já havia sido feito e delinear os passos seguintes. Os casos de teste a seguir verificam se a gramática é capaz de aceitar os exemplos (a maioria deles semalhante aos descritos na seção "Exemplos").

```
import com.google.inject.Inject
import org.eclipse.xtext.testing.InjectWith
import org.eclipse.xtext.testing.XtextRunner
import org.eclipse.xtext.testing.util.ParseHelper
import org.junit.Assert
import org.junit.Test
import org.junit.runner.RunWith
import org.xtext.example.mydsl.cpp.Model

@RunWith(XtextRunner)
@InjectWith(CppInjectorProvider)
class CppParsingTest {
        @Inject
        ParseHelper<Model> parseHelper

        @Test
        def void structBasico() {
                val result = parseHelper.parse('''
                        struct teste {};
                ''')
                Assert.assertNotNull(result)
                Assert.assertTrue(result.eResource.errors.isEmpty)
        }

        @Test
        def void structComVariaveis() {
                val result = parseHelper.parse('''
                        struct Person {
                            char name[50];
                            int age;
                            float salary;
                        };
                ''')
                Assert.assertNotNull(result)
```

```
        Assert.assertTrue(result.eResource.errors.isEmpty)
    }

    @Test
    def void declaraObjetos() {
        val result = parseHelper.parse('''
            struct product {
                int weight;
                double price;
            } apple, banana, melon;
        ''')
        Assert.assertNotNull(result)
        Assert.assertTrue(result.eResource.errors.isEmpty)
    }

    @Test
    def void variavelTipoStruct() {
        val result = parseHelper.parse('''
            struct movies_t {
                string title;
                int year;
            } films [3];

            struct friends_t {
                char name[25];
                string email;
                movies_t favorite_movie;
            } charlie, maria;
        ''')
        Assert.assertNotNull(result)
        Assert.assertTrue(result.eResource.errors.isEmpty)
    }

    @Test
    def void structHeranca() {
        val result = parseHelper.parse('''
            struct A { };
            struct B : A { };
            struct C : B { };
        ''')
        Assert.assertNotNull(result)
        Assert.assertTrue(result.eResource.errors.isEmpty)
    }

}
```

Conclusão

A partir do desenvolvimento do trabalho passamos a conhecer mais sobre a contrução de gramáticas e do uso da ferramento Xtext para a geração das mesmas.

Apesar de uma certa complexidade presente no comando *struct* da linguagem C++ conseguimos simplificá-lo de modo a abranger grande parte das implementações do comando e ao mesmo tempo possuir uma gramática simples.

Tal simplificação acabou deixando de fora pontos importantes do comando e da linguagem C++, como o uso de ponteiros, modificadores como *unsigned*, *long*, além da utilização de *union*. Tais recursos poderiam ser implementados em uma futura expansão da gramática.

Referências

[1] Classes and structures (C++ only) - IBM, acesso em (2018, 22 de agosto), `https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzarg/cplr054.htm`

[2] Data structures, acesso em (2018, 22 de agosto), `http://www.cplusplus.com/doc/tutorial/structures/`

[3] Inheritance (C++ only), acesso em (2018, 22 de agosto), `https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbclx01/inher.htm`

[4] C++ Grammar, acesso em (2018, 22 de agosto), `http://www.nongnu.org/hcb/`

[5] struct (C++), acesso em (2018, 22 de agosto), `https://msdn.microsoft.com/en-us/library/64973255.aspx`

[6] The Grammar for ARM C++ with _opt factored out, acesso em (2018, 22 de agosto), `https://www.cs.dartmouth.edu/~mckeeman/cs48/references/cxx.html`