Software-Qualität mit Halstead-Metriken messen

1. Vorbereitungen

Studieren Sie den Abschnitt "Die Halstead-Metriken" des Artikels "Komplexität und Qualität von Software" aus der Zeitschrift MSCoder (ab Seite 41).

2. Aufgabenstellung

2.2. Halstead-Metriken bestimmen

Entwickeln Sie mit ANTLR einen Programmanalysator zur Bestimmung der Halstead-Metriken von **C**-Programmen!

Folgenden Ausgaben soll Ihr Analysator produzieren:

1. Auflistung aller unterschiedlichen Operatoren und Operanden mit deren Häufigkeiten

2. Halstead-Metriken

- **N** : Programmlänge
- **N1** : Anzahl aller Operatoren
- **N2** : Anzahl aller Operanden
- **n** : Vokabulargröße
- **n1** : Anzahl verschiedener Operatoren
- **n2** : Anzahl verschiedener Operanden
- **Volume V** : Volumen des Programms
- **Difficulty D** : Schwierigkeitsgrad ein Programm zu verstehen
- **Effort E** : Implementierungsaufwand

2.3 Halstead-Lexer

Entwickeln Sie mit Hilfe von ANTLR die Klasse `HalsteadLexer` zum Einlesen der C-Quelldateien. Die Methode `nextToken()` liefert bei jedem Aufruf ein Objekt der Klasse `Token`. Der Lexer klassifiziert die Token wie unten aufgelistet in die Tokentypen OPERAND, OPERATOR oder IGNORE. Am Ende der Eingabe liefert der Lexer den Tokentyp EOF.

OPERAND

- IDENTIFIER – alle Identifier, die keine reservierten Wörter sind.
- TYPESPEC – (type specifiers) Reservierte Wörter, die Typen spezifizieren: `bool, char, double, float, int, long, short, signed, unsigned, void`
- CONSTANT – Zeichenkonstanten, numerische oder String-Konstanten.

OPERATOR

- SCSPEC – (storage class specifiers) Reservierte Wörter, die Speicherklassen beschreiben: `auto, extern, inline, register, static, typedef, virtual, mtuable`
- TYPE_QUAL – (type qualifiers) Reservierte Wörter, die Typen qualifizieren: `const, friend, volatile`
- RESERVED – Andere reservierte Wörter der C++ Programmiersprache: `asm, break, case, class, continue, default, delete, while(, else, enum, for(, goto, if(, new, operator, private, protected, public, return, sizeof, struct, switch(, this, union, namespace, using, try, catch, throw, const_cast, static_cast, dynamic_cast, reiznterpret_cast, typeid, template, explicit, true, false, typename`
- OPERATORS – `'!'|'!='|'%'|'%='|'&'|'&&'|'&='|'('|'*'|'*='|'+'|'++'|'+='|','|'-'|'--'|'-='|'->'|'...'|'/'|'/='|'::'|'<'|'<<'|'<<='|'<='|'=='|'>'|'>='|'>>'|'>>='|'?'|'['|'^'|'^='|'{'|'||'|'='|'~'|';'`

IGNORE

- Kommentare
- Geschlossene Klammern: `')', '}', ']'`
- Doppelpunkt: `':'`
- Schlüsselwort: `'do'`
- `#include`-Zeilen

Bemerkungen

- Die geöffneten runden Klammern nach `if`-, `while`-, `for`- und `switch`-Anweisungen, gehören zu dem Lexem der jeweiligen Anweisung und werden nicht als separate Token geliefert.
- Alle geschlossenen Klammern werden als Tokentyp IGNORE klassifiziert.
- Der Doppelpunkt nach einer `case`-Alternative gehört zum `case` und wird ebenfalls nicht noch einmal gezählt.
- Das Token `do` wird als Tokentyp IGNORE klassifiziert.
- `do-while`-Anweisungen können Sie als `while`-Anweisungen zählen.
- `#include`-Zeilen werden als Tokentyp IGNORE klassifiziert und können ignoriert werden.
- Kommentare werden ignoriert.

Lösungshinweis

Interessante Beispiele für lexikalische Regeln enthält der *Fuzzy Java Lexer* im Kapitel 5.8 „filter option“ des ANTLR3-Buches. Die Grammatik enthält Regeln zum Erkennen von Kommentaren und Strings, die Sie als Vorlage für Ihren Lexer verwenden können.

Wenn Sie ANTLR 4 verwenden möchten, sollten Sie das Kapitel 5.5 „Recognizing Common Lexical Structures“ im ANTLR 4-Buch studieren.

Verwendung des Lexers

Das folgende Programm demonstriert die Verwendung des Lexers. Das Programm zerlegt die Eingabe in einen Tokenstrom. Für jedes Token werden Zeilennummer, Spaltennummer, Tokentyp-Code und Lexem ausgegeben. Die Tokentyp-Codes sind in der Klasse `HalsteadLexer` als ganzzahlige Konstanten deklariert.

```
import org.antlr.runtime.*;
public class Test {
  public static void main(String[] args) throws Exception {
    CharStream input = null;
    // Pick an input stream (filename from commandline or stdin)
    if ( args.length>0 ) input = new ANTLRFileStream(args[0]);
    else input = new ANTLRInputStream(System.in);
    HalsteadLexer lex = new HalsteadLexer(input);
    Token t = lex.nextToken();
    while ( t.getType()!= HalsteadLexer.EOF ) {//Token.EOF works as well
      System.out.printf("%2d:%2d Typ-Code: %2d Lexem: %s\n",
```

```
            t.getLine(),
            t.getCharPositionInLine(),
            t.getType(),
            t.getText());
          t = lex.nextToken();
      }
    }
}
```

Wenn Sie den mit ANLTR generierten Java-Code des Lexers in ein Eclipse-Projekt übernehmen möchten, müssen Sie die ANTLR-Bibliothek **antlr-3.2.jar** in Ihr Projekt einbinden!

Test des Programmanalysators

Testen Sie Ihren Analysator an den in der Vorlesung verwendeten Beispielfunktionen `ggt1` u. `ggt2`. Die `do-while`-Anweisung in `ggt2` kann als `while`-Anweisung gezählt werden.

Testen Sie Ihren Analysator anhand des Beispielprogramms aus dem MSCoder-Artikel (Listing 1, S. 37). Verwenden Sie als Testeingabe einmal das gesamte Programm `Beispiel2.c` und dann jeweils separat die Funktionen `main`, `eval1` und `extract`.

Vergleichen Sie die Ausgaben Ihres Analysators mit den im Listing 2 des MSCoder-Artikels dargestellten Ausgaben. Ihre Ergebnisse für `Beispiel2.c` werden von denen des MSCoder-Artikels leicht abweichen, da dort die `#include`-Zeilen nicht vollständig ignoriert werden. Das Doppelkreuz # wird dort als Operator und der `include` -String als Operand gezählt. Sie sollen die `#include`-Zeilen vollständig ignorieren.