Halstead-Lexer zur Berechnung der Halstead-Metriken

1. Vorbereitungen

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

Wiederholen Sie das Studium des Kapitels 5.5 „Recognizing Common Lexical Structures“ im ANTLR 4-Buch.

2. Halstead-Metriken bestimmen

Entwickeln Sie einen Programmanalysator zur Bestimmung der Halstead-Metriken von **C**-Programmen! Wiederverwenden Sie hierzu Ihre Klassen `LinkedListMultiset<E>` bzw. `TreeMultiset<E>` und Ihre Klasse `Halstead<E>` aus dem Praktikum Informatik II.

Folgende Ausgaben soll Ihr Analysator produzieren:

1. Auflistung aller unterschiedlichen Operatoren und Operanden samt 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

3. Der Halstead-Lexer

Entwickeln Sie mit Hilfe von ANTLR4 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.

`HalsteadLexer`.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.

`HalsteadLexer`.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 – `'!'|'!='|'%'|'%='|'&'|'&&'|'&='|'('|'*'|'*='|'+'|'++'|'+='|',|'-'|'--'|'-='|'->'|'...'|'/'|'/='|'::'|'<'|'<<'|'<<='|'<='|'=='|'>'|'>='|'>>'|'>>='|'?'|'['|'^'|'^='|'{'|'||'|'='|'~'|';'`

`HalsteadLexer`.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.
- 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.

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==null || 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();
    }
  }
}
```

Sie müssen die ANTLR-Bibliothek **antlr-4.5.2-complete.jar** in Ihr Projekt einbinden!

4. 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.

Bewertung: Ohne Erweiterungen sind mit einem Vortag max. 5 Punkte erreichbar.