PS4

Zürcher Hochschule für Angewandte Wissenschaften

Arbeitsblatt: PSPP

Name: **Kurznamen:**

Eigene Programmiersprache zweiter Teil

Eigene Programmiersprache

Jetzt wollen wir einen Schritt weiter gehen und den Parser so ergänzen, dass er eine Turing äquivalente Programmiersprache verstehen kann. Diese Programmiersprache soll neben der Auswertung von arithmetischen Ausdrücken auch Bedingungen und Schleifen enthalten.

Die Programmiersprache soll konkret folgender an C angelehnte Grammatik entsprechen:

```
// Grammatik aus letztem Praktikum
expr = term {("+" | "-") term}
term = factor {("*" | "/") factor}
factor = number | ident | "(" expr ")"
assignement = ident "=" expr ";"
statement = assignment
statementSequence = statement {statement}
program = statementSequence

// geänderte und neue Teile
(Achtung: Implementation im Parser von StatementSequenze auch anpassen)
statement = (assignment | returnStatement | if | while | block)
// neu
block = "{" statementSequence "}"
returnStatement = "return" expr ";"
condition = "(" ["!"] expr ")"
if = "if" condition statement [ "else" statement]
while = "while " condition statement
```

ReturnStatement: der Wert der im Austruck berechnet wird, wird zurückgegeben.

Block: eine StatementSequenz die durch "{" und "}" geklammert wird.

Condition: überprüft einen Int-Wert auf dem Stack auf 0 (= false) oder ungleich 0 (= true). Da wir mit Double Werten rechnen, muss der Wert zuerst auf den nächsten ***gerundet*** und nachfolgend noch in einen ***Ganzzahlenwert*** umgewandelt werden.

if: falls die *condition* != 0 ist, wird das nachfolgende Statement ausgeführt sonst übersprungen.

statement: Eine Anweisung ist entweder ein Assignment, ein *if,* ein *whil*e, ein *block* oder ein *return*.

while: Solange die *condition* != 0 ist, wird das nachfolgende Statement ausgeführt (sonst übersprungen)

Aufgabe 1

Zuerst soll die return Anweisung implementiert werden. Die entsprechend Produktion soll in eine Methode umgewandelt werden und in den Parser eingebaut werden.

Nennen Sie Ihre Klasse nun `Program` und Verwenden Sie folgendes Interface.

```
public interface IProgramm {
      double exec (double arg0);
}
```

Testen Sie den Compiler mit folgendem Programm bzw. führen Sie die Unit Tests aus. Dafür wird das ISimpleProgram Interface verwendet, das ohne Aufrufparameter auskommt.

```
m = $arg0 + 22;
return m;
```

Abgabe:

Praktikum: **PS4.1**

Filename: Program.java

Aufgabe 2

Als nächste Aufgabe soll die if Anweisung implementiert werden. Die Syntax der Anweisungen entnehmen Sie wieder der obigen Grammatik. Implementieren Sie weiter folgendes: Falls die Bedingung (condition) nicht erfüllt ist (den Wert 0 hat) soll der else Teil genommen werden.

Folgendes Programm sollte laufen:

```
// wenn nicht 42, dann Antichrist (=666)
m = $arg0 - 42;
if (!m) {
      return 7;
} else {
      return 666;
}
```

Hinweise:

- Bei der zu überprüfenden Bedingung muss noch berücksichtigt werden, dass boolean eine Ganzzahl (=> i32) ist und wir aber mit Fliesskommazahlen rechnen. **Runden** Sie den Ausdruck einfach auf die nächste Ganzzahl
- Die WASM Code Validierung führt i.d.R. keine sog. Kontrollflussanalyse durch. Das heisst, sie kann im obigen Fall nicht überprüfen, ob alle Kontrollflüsse einen Wert zurück geben; die Validierung möchte aber sicherstellen, dass nicht aus Methode "gefallen werden kann". Aus diesem Grund müssen Code Abschnitte, die nicht erreicht werden, mit der UNREACHBLE (BlockInstruction) versehen werden, i.e. es muss am Schluss des Programms noch die UNREACHBLE Instruktion angehängt werden.

Abgabe:

Praktikum: **PS4.2**

Filename: Program.java

Aufgabe 3

In dieser Aufgabe soll die while-Anweisung realisiert werden.

Hinweise:

- Es muss die Block und Loop Anweisung ineinander geschachtelt werden (siehe Folien).

Folgendes Programm sollte nun laufen

Beispiel-Code = Fakultät

```
m = $arg0;
s = 1;
while (m) {
  s = s * m;
  m = m - 1;
}
return s;
```

Abgabe:

Praktikum: **PS4.3**

Filename: Program.java

Aufgabe 4 (optional)

Eine mögliche Erweiterung wäre, dass auch die Vergleichsoperatoren unterstützt werden, so dass z.B. `if (a < 3)` oder `if ( a != 0)` in einer Bedingung verwendet werden könnten.