Zürcher Hochschule
für Angewandte Wissenschaften

Eigene Programmiersprache zweiter Teil

Eigene Programmiersprache

Jetzt wollen wir einen Schritt weiter gehen und den Parser so ergänzen, dass er eine Turing vollständige 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 StatementSequent 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 *while*, ein *block* oder ein *return*.

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

Aufgabe 1

In dieser Aufgabe sollen die return und die if Anweidung realisiert werden. Dafür müssen Sprunganweisungen generiert werden. Die Syntax der Anweisungen entnehmen Sie einfach 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

Resultat: 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 keine Kontrollflussanalyse durch. Das heisst es kann im obigen Fall nicht überprüfen, ob alle Kontrollflüsse einen Wert zurück geben. Aus diesem Grund müssen Code Abschnitte, die nicht erreicht werden können, mit der UNREACHBLE (BlockInstruction) gekennzeichnet werden. Konkret muss am Schluss der obigen Funktion noch so eine Instruktion angehängt werden. Alternativ kann am Schluss einer Funktion mit Rückgabe Wert ein (Dummy-)Wert vom Rückgabe-Typ auf den Stack plaziert werden und eine Return Anweisung folgen. Dies auch, wenn besagte Code-Sequenz nie ausgeführt wird.

Aufgabe 2

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;
```

Aufgabe 3 (optional)

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