Übersetzerbau 1
WS11/12

Prüfer: Prof. Dr. Michael Philippsen, Dipl.-Inf. Stefan Kempf
Ergebnis: 1.7
Die Fragen hat eigentlich nur Stefan gestellt. Herr Philippsen hat viele Kommentare gemacht. Insgesamt eine sehr entspannte Prüfung in netter Atmosphäre. Ich habe in jeder Aufgabe kleine Fehler gemacht und wurde jedes Mal wieder mit einem kleinen Hinweis darauf aufmerksam gemacht. Stift und Papier ist vorhanden und sollte genutzt werden. In Anlehnung an ein anderes Protokoll habe ich „P“ für „Prüfer“ und „S“ für „Student“ gewählt. Dieses Protokoll ist nach bestem Wissen und Gewissen angefertigt, erhebt aber keinen Anspruch auf Vollständigkeit oder Richtigkeit.
Gegeben ist ein Codestück in Pseudocode, in etwa wie folgt:

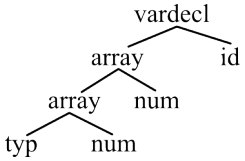
```
float a = 3.12;
int b = 5;
int[10][20] C;

int main(){
    int z = 3;
    int y = C[z][a + b + 3];
    return y;
}
```

P: Wie übersetzt ein Compiler dieses Programm?

S: Erst werden vom Lexer Token erzeugt. Der Parser macht dann daraus einen Syntaxbaum.

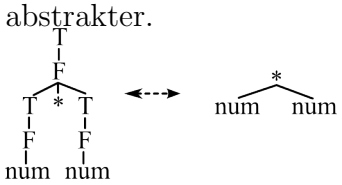
P: Wie sieht denn der Baum der Arraydeklaration aus?

S:



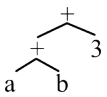
P: Wie ist das dann mit AST und dem, was direkt beim Parsen aufgebaut wird?

S: Zum Beispiel um Punkt vor Strich umzusetzen wird in der Grammatik zwischen Term und Faktor unterschieden. Das braucht man dann im AST nicht mehr. Links ist als ein konkreter Syntaxbaum, rechts ein abstrakter.



P: Wie sieht dann der Baum für das $a + b + 3$ in dem Arrayzugriff aus?

S:



P: Wie wird der dann nach dem Parser weiterverwendet?

S: Dann kommt die semantische Analyse. Da wird z.B. einmal über den Baum gegangen und die Typen gefprüft (mit Definitionstabelle).(Hier war dann noch gefragt über den Teilbaum zu gehen, festzustellen, dass a ein float ist und dass der Arrayzugriff damit nicht ganz richtig ist. Dabei auch gleich Erklärung von Prototypen.)

P: Sind noch irgendwelche Fehler in dem Programm?

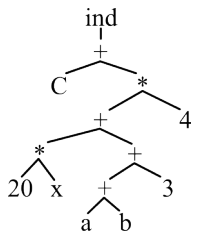
S: *starrt auf das Programm* ich sehe keine mehr.

P: Es sind auch keine mehr da. Bei Graham & Glanville werden auch Bäume benutzt. Wie funktioniert das?

S: Der entsprechende Baum wird in Präfixschreibweise geschrieben und dann geparst. Dabei werden die zugehörigen Maschinenbefehle erzeugt.

P: Zeichnen Sie bitte den Baum des Arrayzugriffs.

S:



P: Schreiben Sie den Teil ab dem zweiten Plus in Präfixnotation.

S: +*20x++ab3

P: Wie geht es jetzt weiter? Sie haben eine Maschine mit Register-Register und Register-Immedeate Ausdrücken. Die Ergebnisse werden immer in Registern gehalten.

S: $R \leftarrow op\ R\ R$
$R \leftarrow op\ R\ const$
$R \leftarrow const$

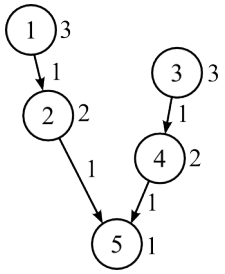
P: Parsen Sie den Baum. Gehen Sie davon aus, dass die Variablen bereits in Registern sind.

S: LR Parsen wie auf den Folien beschrieben. Dabei Maschinencode erzeugen. Ich habe hier Pseudoassembler benutzt mit dem Ziel ganz links. Es wären auch einfache Zuweisungen möglich gewesen.

```
MOV R1, 20
MUL R2, R1, x
ADD R3, a, b
ADD R4, R3, 3
ADD R5, R2, R4
```

P: Wenn wir die Instruktionen in der Reihenfolge ausführen, dann müssen manche erst auf das Ergebnis der Vorgängerinstruktion warten. Wie können wir Umstellen, um diese Wartezeiten zu vermeiden? Dabei soll die Ausführung einer Instruktion einen Zyklus brauchen und die Wartezeit genauso lange.

S: Da können wir einen Graphen aufstellen und Delays berechnen.



(Die Nummern in den Kreisen sind die Zeilennummern aus dem Beispiel davor. Die daneben sind die zugehörigen Delays)

P: In welcher Reihenfolge werden die jetzt angeordnet?

S: Wir fangen mit den Instruktionen an, die keine Voraussetzungen haben, also 1 und 3 und nehmen dann das mit dem höchsten Delay.