

Compilerbau SS 2019 - Aufgabenblatt 1

Aufgabe 1 (Grundbegriffe)

Eignen Sie sich das im Kapitel 2 des Skripts enthaltene programmiersprachliche Grundwissen an.

Aufgabe 2 (Phasen eines C-Compilers und Repräsentationen eines Programms)

Erstellen Sie eine Datei `aufgabe1.c` mit folgendem Inhalt:

```
#include <stdio.h>
#define ZAHL 5

int fak(int n){
    int n1=n;
    if (n1<2)
        return 1;
    else
        return n1*fak(n1-1);
}

int main(){
    printf("fak(ZAHL)=%d\n", fak(ZAHL));
}
```

- a) Benutzen Sie den GNU-C-Compiler *gcc*, um aus dem Programmquelltext ein ausführbares Maschinenprogramm zu erzeugen:

```
gcc aufgabe1.c
```

Finden Sie heraus, wo das ausführbare Maschinenprogramm nach dem Compileraufruf gespeichert ist und wie man es aufruft.

- b) Die Verarbeitung erfolgt in mehreren Stufen, die im obigen Fall alle zusammengefasst sind:

Stufe	Eingabedatei	Ausgabedatei
C-Präprozessor	<code>aufgabe1.c</code>	<code>aufgabe1.i</code>
C-Compiler	<code>aufgabe1.i</code>	<code>aufgabe1.s</code>
Assembler	<code>aufgabe1.s</code>	<code>aufgabe1.o</code>
Linker	<code>aufgabe1.o</code>	<code>a.out</code>

(Die Zwischenergebnisse der einzelnen Stufen werden wieder gelöscht, wenn man nicht *gcc* mit „-save-temps“ aufruft.)

Überlegen Sie, welche Aktionen der Präprozessor in diesem Beispiel durchführt und wie das Ergebnis wohl aussieht. Überprüfen Sie ihre Überlegung, indem Sie sich die Präprozessorausgabe des GNU-C-Compilers *gcc* ansehen:

```
gcc -E aufgabe1.c -o aufgabe1.i
```

In der Datei `aufgabe1.i` steht jetzt die Präprozessorausgabe. Finden Sie heraus, wieviele Dateien der Präprozessor dabei zusammenführt. Prüfen Sie, ob die Präprozessorkonstante `ZAHL` überall durch `5` ersetzt ist.

- c) Der eigentliche C-Compiler erzeugt aus der Präprozessorausgabe eine Assemblerdatei `aufgabe1.s`:

```
gcc -S aufgabe1.i
```

Betrachten Sie die Assembler-Datei, insbesondere den Code der Funktion `fak`.

- d) Der gcc-Assembler erzeugt danach ein sogenanntes „Objektmodul“ in der Datei `aufgabe1.o`

```
gcc -c aufgabe1.s
```

Dieses wird vom Linker zu einem ausführbaren Programm `a.out` vervollständigt:

```
gcc -o aufgabe1 aufgabe1.o
```

Überlegen Sie, was wohl der wesentliche Unterschied zwischen `aufgabe1.o` und `a.out` ist.

- e) Das Programm besteht nur aus einem Modul. Wie könnte man es in zwei Module zerlegen? Zerlegen Sie das Programm in zwei Module und übersetzen Sie die Module getrennt mit `gcc`. Benutzen Sie danach wieder den Linker, um ein ausführbares Programm zu erzeugen.

Aufgabe 3 (Lexikalische Analyse)

Der eigentliche C-Compiler muss die Struktur des Quelltexts erkennen und dazu zunächst die lexikalischen „Tokens“ bestimmen. Tun Sie dies für die Definition der Funktion „main“ aus dem obigen C-Beispielprogramm. Geben Sie zu jedem Token das „Lexem“ und die Tokenkategorie an.

Aufgabe 4 (Ausdrücke)

- a) Wer legt fest, wie Ausdrücke in einer Programmiersprache aussehen? Wie legt man das fest?
- b) Wann und in welcher Weise wird ein Ausdruck ausgewertet?
- c) Betrachten Sie die nachfolgenden Java-Ausdrücke. Überlegen Sie sich, in welcher Weise diese Ausdrücke verarbeitet werden müssen: Welche Prüfungen müssen durchgeführt werden? Welche Informationen müssen berechnet werden? In welcher Weise, zu welchem Zeitpunkt und von welchem Programm werden die Werte berechnet?

- (1) `3*(4+6)*7`
- (2) `3+7.4`
- (3) `3+i`
- (4) `3+f(5)`