Übungsblatt 4 – Ausgabe am 18.05.2020

Abgabe bis 25.05.2020, 12 Uhr, per Mail

Aufgabe 4.1: Präprozessor-Programmierung.

Es soll eine konfigurierbare Funktion *optimum* implementiert werden. Die Funktion hat zwei Zahlen *x* und *y* als Eingabeparameter und gibt das Optimum dieser beiden Zahlen zurück. Die Funktion soll zwei unabhängig konfigurierbare Variabilitätsdimensionen aufweisen:

- Schnittstellenvariabilität: Die Funktion soll entweder Zahlen vom Typ *int* (Feature „Int“) oder vom Typ *float* (Feature „Float“) verarbeiten.
- Funktionalitätsvariabilität: Die Funktion soll entweder die *maximale* (Feature „Max“) oder die *minimale* (Feature „Min“) der beiden Zahlen ermitteln und zurückgeben.

a) Geben Sie eine Implementierung der Funktion *optimum* mit möglichst feingranularer Variabilität mithilfe von Präprozessor-Direktiven ein.
b) Ist eine direkte Implementierung Ihrer Lösung aus a) durch Variabilitätskodierung möglich? Begründen Sie und geben Sie eine mögliche Lösung an.

Hinweise: Sie können C-Syntax oder Java-Syntax für Ihre Lösung verwenden. Eine Kodierung des zugehörigen Feature-Modells können Sie als gegeben voraussetzen.

Aufgabe 4.2: Präprozessor-Programmierung.

Geben Sie für die folgenden Code-Schnipsel die jeweilige Ausgabe des C-Präprozessors an und erläutern Sie die Ausgabe kurz in einem Satz. Recherchieren Sie dazu gegebenenfalls in der CPP-Dokumentation.
Hinweis: Sie können die Aufgabe entweder „auf Papier“ lösen oder, falls vorhanden, gerne auch den C-Präprozessor eines bei Ihnen installierten C-Compilers einzeln aufrufen (schauen Sie dafür in die Dokumentation).

a)
```
foo = X;
#define X 4
bar = X;
```

b)
```
/*
*/ # /*
*/ defi\
ne FO\
O 10\
20
bar = FOO;
```

c)
```
#define min(X,Y) ((X) < (Y) ? (X) : (Y))
min (1,2)
min (min (1,2),2)
```

d)
```
#define COMMAND(NAME) { #NAME, NAME ## _command}
struct command commmands[] = {
        COMMAND (quit),
        COMMAND (help),
        ...
};
```

e)
```
 #define double(x) (2*(x))
#define call_with_1(x) x(1)
call_with_1 (double)
```

f)
```
#define CALLFOO(X) foo(X,
x = CALLFOO(1) 2);
```

g)
```
#define foo (4 + foo)
foo
```