BERGISCHE UNIVERSITÄT
WUPPERTAL
GAUSS-STRASSE 20
42119 WUPPERTAL
TELEFAX (0202) 439-2901
TELEFON (0202) 439-0
WWW www.uni-wuppertal.de

Fachbereich C

MATHEMATIK UND NATURWISSENSCHAFTEN
Fachgruppe Mathematik und Informatik

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Hans-Juergen.Buhl@math.uni-wuppertal.de
WWW: http://www.math.uni-wuppertal.de/∼buhl

Programming by Contract

WS 2005/2006 – Übungsblatt 7

Ausgabe: 8. Dezember 2005

Abgabe: bis spätestens 15. Dezember 2005
in der Vorlesung
oder per email an `c.markmann@uni-wuppertal.de`

Aufgabe 1. *Integer-Overflow*

Schreiben Sie in C++ eine Funktion

```
int fakultaet(int n)
```

und testen Sie diese. Für welche Argumentwerte ist das Ergebnis befriedigend? Was geschieht bei größeren Argumentwerten, was bei negativen? Warum?

Schreiben Sie einen Contract für diese erste Version (Vor- und Nachbedingungen).

Ändern Sie die Funktion dann so ab (Abfrage und Verzweigung in Abhängigkeit des Argumentwertes), dass sie nur noch richtige Ergebniswerte liefert *oder* eine Exception auslöst. Testen Sie erneut und modifizieren Sie Vor- und Nachbedingung entsprechend.

Schreiben Sie als letzte (unabhängig vom Wert von `INT_MAX` vollständig portable) Version, die vor jeder Multiplikation testet, ob `INT_MAX` überschritten würde und in diesem Fall die Exception auslöst.

Warum sollte man den Datentyp des Resultats, des Arguments und der Hilfsvariablen der Funktion nicht als `unsigned int` deklarieren?

Benutzen Sie statt des Datentyps `int` den Typ `long`. Wie groß ist jetzt der größte erfolgversprechende Argumentwert?

Aufgabe 2. *Hoare-Tripel*

Schreiben Sie jetzt für die letzte Version der Funktion `fakultaet()` (Aufgabe 1) für jede Anweisung das Hoare-Tripel nieder und verifizieren Sie dann, dass Ihre Funktion das richtige Ergebnisse liefert.

Dazu benötigen Sie den Contract der Multiplikationsfunktion

`*` : `int` $\times$ `int` $\rightarrow$ `int`

Formulieren Sie also die richtigen Vor- und Nachbedingungen.

Aufgabe 3. *Integer-Overflow-Vermeidung*

Wie sehen die entsprechenden Fallunterscheidungen zur Vermeidung des `int`-Overflows (beziehungsweise -Underflows) bei den vier Grundrechenarten und bei einer selbst geschriebenen Funktion

`int square(int)` aus?

Welche anderen Konstanten aus `limits.h/math.h/values.h` sind für ähnliche Anwendungen sinnvoll einsetzbar (z.B. zur Vermeidung eines zwischenzeitlichen Auftretens des Wertes Unendlich?

Aufgabe 4. *isnan()*

Schildern Sie Anwendungen für analoge Algorithmenteile mit Hilfe von

`isinf()`, `isnan()`, `finite()`, `isnormal()`, `isunordered()`.

Welche Vorteile hat die Funktion `isgreater()` gegenüber dem Operator $>$?

Aufgabe 5. *unordered values*

Notieren Sie die Hoare-Tripel zu jeder Anweisung der beiden Varianten des Algorithmus der Seite

http://www.math.uni-wuppertal.de/~buhl/teach/exercises/PbC0506/unorderedFloat.pdf

und begründen Sie damit die Unterschiede der Plots der beiden Varianten.