Prüfung Informatik D-MATH/D-PHYS 10. 3. 2006

Dr. Bernd Gärtner

Lösung.

Aufgabe 1.

(a)
```
(2e1+((3/6)/2))-7 →
(2e1+(0/2))-7 →
(2e1+0)-7 →
2e1-7 →
13.0.
```

(b)
```
(int(3.25f*6)!=(38./2))||((((-8)-1.5)>3)&&(1.1==(110/100.0f))) →
(int(3.25f*6)!=19.0)||((((-8)-1.5)>3)&&(1.1==(110/100.0f))) →
(int(19.5f)!=19.0)||((((-8)-1.5)>3)&&(1.1==(110/100.0f))) →
(19!=19.0)||((((-8)-1.5)>3)&&(1.1==(110/100.0f))) →
false||((((-8)-1.5)>3)&&(1.1==(110/100.0f))) →
(((-8)-1.5)>3)&&(1.1==(110/100.0f)) →
(-9.5>3)&&(1.1==(110/100.0f)) →
false&&(1.1==(110/100.0f)) →
false.
```

(c)
```
(2-(3*0.75))<(-(1/4))||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
(2-2.25)<(-(1/4))||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
-0.25<(-(1/4))||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
-0.25<(-0)||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
-0.25<0||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
true||(((3.159/143)>0.22)&&(7-8)>(-0.9)) →
true.
```

Aufgabe 2.

```
unsigned int cross_sum(unsigned int n)
{
unsigned int c = 0;
while (n > 0) {
c += n % 10;
n /= 10;
}
return c;
}

unsigned int iterated_cross_sum(unsigned int n)
{
```

```
unsigned int c = cross_sum(n);
if (c < 10)
return c;
else
return iterated_cross_sum(c);
}
```

Aufgabe 3. Hier ist folgende Tabelle hilfreich.

(b1,b2)	(b2,!b1)
(true,true)	(true, false)
(true,false)	(false, false)
(false,false)	(false, true)
(false, true)	(true,true)

Wir sehen also ein rotierendes Muster; bei jedem rekursiven Aufruf wird das Paar bestehend aus zweitem und drittem Aufrufparameter in der obigen Tabelle eine Zeile "weitergeschaltet" und hat für `n-4` schliesslich wieder (`true`,`true`) erreicht. Die Nachbedingung ist demnach die folgende:

```
POST: Rueckgabewert ist true genau dann, wenn n durch 4 teilbar ist
```

Aufgabe 4. .

(a) `a != 4 - a && !(a < 3 * a - 7 || a >= a * a * a)` $\Longleftrightarrow$ *De Morgan*
`a != 4 - a && a >= 3 * a - 7 && a < a * a * a` $\Longleftrightarrow$ *einf. Umformung*
`a != 2 && 7 >= 2 * a && a < a * a * a` $\Longleftrightarrow$ $a \in \mathbb{Z}$
`a != 2 && 3 >= a && a < a * a * a` $\Longleftrightarrow$ $a<a3 \Longleftrightarrow a>1$ *für* $a \in \mathbb{Z}$
`a == 3`.

(b) `b < 3 || a - 1 / (b - 2) < 5 / a + 1 && a >= b` $\Longleftrightarrow$ *redundante Ergänzung*
`b < 3 || b >= 3 && a - 1 / (b - 2) < 5 / a + 1 && a >= b`.
Für $b \geq 3$ und $a \geq b$ gilt `a - 1 / (b - 2)` $\geq$ `a - 1` sowie `5 / a + 1` $\leq$ 2.
Daraus folgt
`b >= 3 && a - 1 / (b - 2) < 5 / a + 1 && a >= b` $\Longrightarrow$
`b >= 3 && a - 1 < 2 && a >= b` $\Longrightarrow$
`false`.
Demnach ist das ursprüngliche Prädikat äquivalent zu `b < 3`.

Aufgabe 5. Hier sind die 6 Fehler.

- Zeile 2: Klammern fehlen hinter `main` (syntaktischer Fehler)
- Zeile 5: *expression* `++i` fehlt (semantischer Fehler, führt zu Endlosschleife)
- Zeile 6,7,10: `std::` fehlt vor `cout`, `cin` (syntaktischer Fehler)

- Zeile 7: `n` nicht definiert (syntaktischer Fehler)
- Zeile 8: Diese Zeile ist nicht im Scope der Funktion `update_max` (syntaktischer Fehler)
- Zeile 14: Die Funktion `update_max` hat keinen Effekt, da der Parameter `max` kein Referenzparameter ist (semantischer Fehler, Maximum bleibt stets bei -1)

Hier ist die korrigiert Version des Programms.

```
#include<iostream>

void update_max (int& max, int n)
{
if (n > max) max = n;
}

int main ()
{
int max = -1;
for (unsigned int i=0; i<5; ++i) {
unsigned int n;
std::cout << "Next natural number =? ";
std::cin >> n;
update_max(max, n);
}
std::cout << "Maximum is " << max << "\n";
return 0;
}
```

Aufgabe 6.

```
class Unsigned {
public:
// POST: *this wurde mit x initialisiert
Unsigned (unsigned int x);

// POST: y wurde zu *this addiert
Unsigned& operator+=(cosnt Unsigned& y);

// POST: Rueckgabewert ist unsigned-int Wert von *this
unsigned int value() const;

// POST: Ruckgabewert ist true genau dann, wenn *this
// nicht uebergelaufen ist, d.h. wenn der berechnete
```

```
// Wert dem mathematisch korrekten Wert entspricht
bool is_exact() const;

private:
unsigned int val; // represented value
bool ok; // overflow flag
};

Unsigned::Unsigned(unsigned int x)
: val(x), ok(true)
{}

// x += y ok <=> x,y ok and new x >= y
Unsigned& Unsigned::operator+=(const Unsigned& y)
{
val += y.val;
ok = ok && y.ok && val >= y.val;
return *this;
}

unsigned int Unsigned::value() const
{
return val;
}

bool Unsigned::is_exact() const
{
return ok;
}
```