Informatik 1 für Regenerative Energien

Klausur vom 9. März 2011: Lösungen

Jörn Loviscach

Versionsstand: 15. März 2011, 14:57

1. 7 ist 00000111_2 und -9 ist 11110111_2.

```
  00000111
 +11110111
       111
 ---------
  11111110
```

2. `~a` ist `0xB0`, `a|b` ist `0xCF`, `a&b` ist `0x43`.

3.
```
(((3.0f + (b*2.0f)) < 5.0f) || (a && ((b - (101%2)) >= 0.0f)))
          __84.0f_                            ____1____
  ________87.0f______                    _____41.0f_____
 _____________false____________         _____________true__________
                                   ________________true________________
_________________________________true_________________________________
```

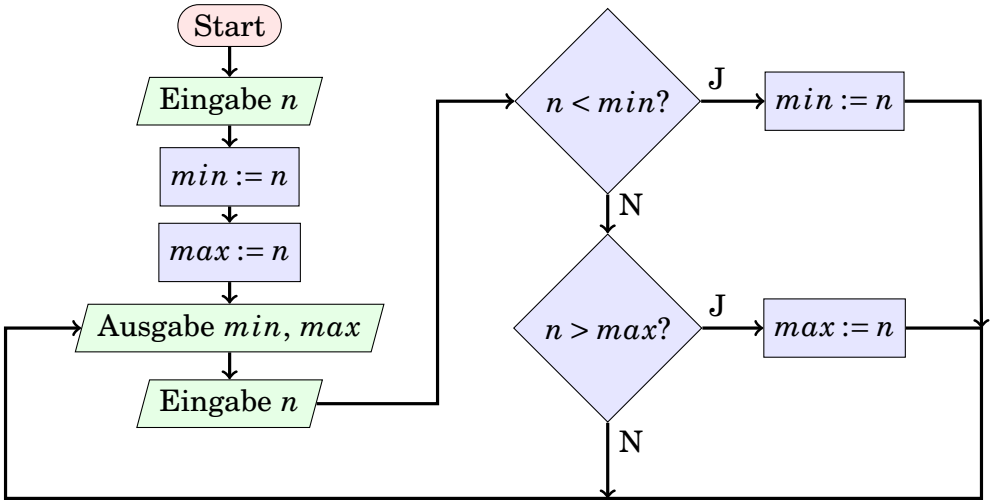
4.
```
int maximum(int a, int b, int c) // Rückggabetyp int statt void
{
  int result = a; // int zur Deklaration/Definition fehlte
  if(b > result)
  {
    result = b;
  }
  if(c > result)
  {
    result = c;
  }
  return result; // Rückgabewert fehlte
}
```

5.
```
#include <math.h>
// ...
double length(Vector a)
{
```

```
  return sqrt(a.x*a.x + a.y*a.y);
}
```

6.
```
int f(int x, int y)
{
  return 3*x + 4*y;
}
// ...
int a = 13;
int b = 42;
int c = f(a, b);
int d = f(b, c);
```

7.



8.
```
int countIdentical(char a[], char b[])
{
  int i = 0; // Um mit extrem langen Zeichenketten umzugehen,
             // wäre size_t statt int sicherer,
             // aber das hatten wir nicht.
  while(a[i] != 0 && b[i] != 0 && a[i] == b[i])
  {
    i++;
  }
  return i;
}
```

Oder aufwendiger:

```
int countIdentical(char a[], char b[])
{
  int lengthA = strlen(a); // besser size_t
  int lengthB = strlen(b);
  int i = 0; // besser size_t
  for(; i < lengthA && i < lengthB; i++)
  {
    if(a[i] != b[i])
```

```
    {
      break;
    }
  }
  return i;
}
```

9. Ab einem Wert von 256 für `x`, weil $256 \cdot 256 = 2^{16}$ nicht mehr in 16 Bit passt, denn die größte unsigned-Zahl ist dort $2^{16} - 1$.

10.
```
bool containsDuplicates(int numbers[], int count)
{
  for(int i = 0; i < count-1; i++) // letzter unnötig
  {
    for(int j = i+1; j < count; j++)
    {
      if(numbers[i] == numbers[j])
      {
        return true;
      }
    }
  }
  return false;
}
```

    Oder nicht so effizient:

```
bool containsDuplicates(int numbers[], int count)
{
  for(int i = 0; i < count; i++)
  {
    for(int j = 0; j < count; j++)
    {
      if(j == i)
      {
        continue;
      }
      if(numbers[i] == numbers[j])
      {
        return true;
      }
    }
  }
  return false;
}
```

11. n Zahlen werden mit jeweils $n - 1$ anderen verglichen, macht $n(n - 1)$ Vergleiche. (Wenn man das geschickt macht, hat man nur die Hälfte davon, weil aus $x = y$ folgt, dass $y = x$. Das ist zwar doppelt so schnell; der Faktor 1/2

ändert aber die asymptotische Laufzeit nicht.) Die asymptotische Laufzeit ist $O(n(n-1)) = O(n^2 - n) = O(n^2)$. (Das n wird asymptotisch gegenüber dem n^2 unbedeutend.)

12. Zum Beispiel:

```
unsigned int f(unsigned int x)
{
  return x/2;
}
```