Tentamen Imperatief Programmeren

maandag 8 november 2010, 9:00-12:00

- Schrijf boven ieder blad je naam, studentnummer, studierichting en volgnummer van het blad. Schrijf op het eerste blad het totaal aantal ingeleverde bladen.
- Je kunt in totaal 100 punten verdienen. De eerste 10 punten krijg je cadeau. Bij iedere opgave staat zijn puntenwaardering vermeld.
- Lees eerst een opgave volledig door alvorens deze te maken.
- Schrijf netjes en zorgvuldig met een pen (geen potlood).
- Je hebt 3 uur de tijd. Gebruik deze nuttig. Als je snel klaar bent, gebruik dan de resterende tijd om je antwoorden nog eens te controleren.
- Succes!

Opgave 1: Toekenningen (20 punten)
Bepaal voor ieder van de onderstaande annotaties de keuze die op de plaats van de lege regel (.....) ingevuld kan worden. Per onderdeel is er precies één keuze mogelijk. De variabelen `x`, `y` en `h` zijn van het type `int`. Let erop dat `X` en `Y` (met hoofdletter!) specificatie-constanten (en dus geen variabelen) zijn.

1.1
```
/* x + 5 == X */
.....
/* x == 3*X + 1 */
```

(a) `x = 3*x + 12;`
(b) `x = 3*x + 14;`
(c) `x = 3*x + 16;`

1.2
```
/* 4*x + 5*y == X */
.....
/* x + y == X */
```

(a) `x = 4*x + 4*y;`
(b) `x = 3*x + 4*y;`
(c) `x = -3*x - 4*y;`

1.3
```
/* x + Y == X, y + z == Y */
x = x + y; y = y - z;
.....
```

(a) `/* x + z == X, y + 2*z == Y */`
(b) `/* x == X, y == Y */`
(c) `/* x + y + Y == X, x + y - z == Y */`

1.4
```
/* x == 2*X, y == X + Y */
x = x - y; y = y - x;
.....
```

(a) `/* x == Y - X, y == 2*Y - X */`
(b) `/* x == X - Y, y == 2*Y */`
(c) `/* x == X - Y, y == 2*X */`

1.5
```
/* x == Y, y == X */
x = x + y; y = x - y; x = x + y;
.....
```

(a) `/* x == Y, y == X */`
(b) `/* x == Y + 2*X, y == X */`
(c) `/* x == X + 2*Y, y == Y */`

1.6
```
/* x == X + Y, y == 2*X + 3*Y */
x = x - y; y = x + y; x = y - x;
.....
```

(a) `/* x == 3*X + 2*Y, y == X - Y */`
(b) `/* x == 2*X + 3*Y, y == X + Y */`
(c) `/* x == 3*X + Y, y == X + Y */`

Opgave 2: Zoek de 5 fouten (10 punten)

Het onderstaande programma is een rekenhulpje voor geheeltallig worteltrekken. De geheeltallige wortel van een positieve integer `n` is de grootste integer `m` zodanig dat `m*m <= n`. Het programma bevat 5 fouten. Geef van iedere fout het regelnummer en geef een correctie.

```
1 #include <stdio.h>
2
3 void wortel(int n) {
4   int p = 1;
5   int q = n+1;
6   while (p + 1 != q) {
7     /* invariant: 1 <= p*p <= n < q*q */
8     int mid = (p + q)/2;
9     if (mid*mid < n) {
10      p = mid;
11    } else {
12      q = mid;
13    }
14  }
15  return q;
16 }
17
18 int main(int argc, char *argv[]) {
19   do {
20     printf("Geef een geheel getal (<= 0 is stoppen): ");
21     scanf("%d", &n);
22     if (n > 0) {
23       printf("De geheeltallige wortel van %d is %d\n", n, wortel());
24     }
25   } while (n > 0);
26
27   return 0;
28 }
```

Opgave 3: Tijdscomplexiteit (20 punten)
In deze opgave is `N` een natuurlijk getal, met `N>0`. Geef van ieder van de volgende programmafragmenten aan wat de scherpste bovengrens is voor het aantal rekenstappen dat het fragment uitvoert in termen van `N`. Een algoritme dat N stappen doet is dus $O(N)$ en niet $O(N^2)$ omdat $O(N)$ de scherpste bovengrens is.

1. ```
   int i, fac = 1;
   for(i = 1; i < N; i++) {
     fac *= i;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2. ```
   int i, prod = 1;
   for (i = 0; i < N; i+=2) {
     prod *= i;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.
```
int i, som = 0;
for (i = N; i != 0; i /= 10) {
  som += i;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int i = 1;
while (2*i <= N*N) {
  i++;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

5.
```
int i, j, prod = 1;
for (i = 1; i < N; i++) {
  for (j = 1; j < N; j*=2) {
    prod = prod*j;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6.
```
int i = 0, som=0, prod=1;
while (som < N) {
  i++;
  prod = prod * i;
  som += i;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Opgave 4: Logaritme (20 punten)
Ieder geheel getal n groter dan 1 kan uniek geschreven worden als een product van zijn priemfactoren; de zogenaamde priemontbinding van n. Bijvoorbeeld, de priemontbinding van het getal 155232 is `2*2*2*2*2*3*3*7*7*11`.

De logaritme heeft de volgende eigenschap: `log(a*b) = log(a) + log(b)`, waarbij `a>0` en `b>0`.

Dit betekent dus dat het volgende geldt:
`log(155232)=log(2)+log(2)+log(2)+log(2)+log(2)+log(3)+log(3)+log(7)+log(7)+log(11)`.

Natuurlijk is het veel leesbaarder als we zouden schrijven:
`log(155232)=5*log(2)+2*log(3)+2*log(7)+log(11)`

Schrijf een functie `toonLogOntbinding(int n)` die de bovenstaande ontbinding van de logaritme van `n` afdrukt. Dus de aanroep `toonLogOntbinding(155232)` dient af te drukken:
`log(155232)=5*log(2)+2*log(3)+2*log(7)+log(11)`

Opgave 5: Kakuro sommen (20 punten)

Een populaire puzzel is de zogenaamde Kakuro. Een Kakuro lijkt op een kruiswoordpuzzel, de witte vakjes moeten worden ingevuld. De gezochte 'woorden' bestaan echter niet uit letters maar uit cijfers, vandaar dat ze 'sommen' worden genoemd. Voor een som mogen de cijfers van 1 tot en met 9 worden gebruikt. Een cijfer mag niet vaker dan één keer in een som voorkomen.

Kakuro-sommen zijn sommen waar maar één mogelijke combinatie van getallen geldig is. Een voorbeeld van een kakuro-som is "4 in 2". Hiermee wordt bedoeld dat de som 4 gemaakt moet worden, waarbij de som bestaat uit twee termen. Voor de som 4 kan dit inderdaad precies op één manier, namelijk $4 = 1 + 3$. Natuurlijk beschouwen we de de sommen $4 = 1 + 3$ en $4 = 3 + 1$ als dezelfde som. Merk op dat $2 + 2$ niet mogelijk is vanwege vanwege de eis dat ieder cijfer in een som ten hoogste één maal voor mag komen. Een langere kakuro-som is 42 in 8, want 42=1+2+4+5+6+7+8+9. Er bestaat geen andere manier om 42 te schrijven als een som van 8 verschillende cijfers. Het getal 12 is geen kakurosom van het type "12 in 4", immers 12=1+2+3+6 maar ook 12=1+2+4+5 .

Schrijf een functie `isKakuroSom(int som, int n)` die bepaalt of het getal `som` een kakuro-som in `n` is. Als dit het geval is, dan moet de functie 1 retourneren. Zo niet, dan retourneert de functie 0.

[Hint: het is handig om een recursieve hulpfunctie te maken die bepaalt op hoeveel manieren het getal `som` als een som van `n` verschillende cijfers geschreven kan worden.]