Tentamen Imperatief Programmeren

8 november 2012, 14:00-17:00

- 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 dient te worden. Per onderdeel is er precies één keuze mogelijk. De variabelen `x`, `y` en `h` zijn van het type `int`. Let erop dat `X`, `Y` en `Z` (met hoofdletter!) specificatie-constanten (en dus geen variabelen) zijn.

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

```
(a) x = x/3 - 1;
(b) x = 3*x + 1;
(c) x = (x - 1)/3;
```

1.2
```
/* x == 3*X + 1 */
.....
/* x == X */
```

```
(a) x = x/3 - 1;
(b) x = 3*x + 1;
(c) x = (x - 1)/3;
```

1.3
```
/* x == y*y + X, y + 1 == Y */
.....
/* x == y*y + X, y == Y */
```

```
(a) y = y - 1; x = x + 2*y - 1;
(b) y = y + 1; x = x + 2*y - 1;
(c) x = x + 2*y - 1; y = y - 1;
```

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

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

1.5
```
/* x == X + Y, y == X + Z, z == Y + Z */
y = (y + z - x)/2; z = x - z + y; x = x - z;
.....
```

```
(a) /* x == X, y == Y, z == Z */
(b) /* x == X, y == Z, z == Y */
(c) /* x == Y, y == Z, z == X */
```

1.6
```
/* x == X + Y + Z, y == Y, z == Z */
y = x - y - z; z = x - y - z; x = x - y - z;
.....
```

```
(a) /* x == Y, y == Z, z == X */
(b) /* x == Z, y == X, z == Y */
(c) /* x == X, y == Y, z == Z */
```

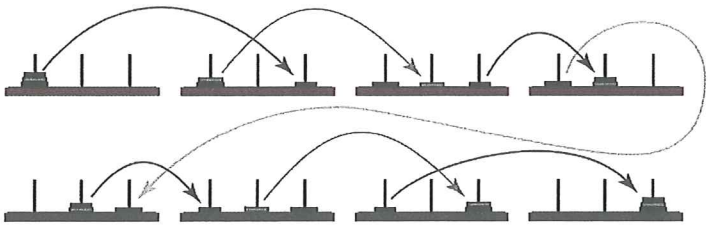
Opgave 2: Zoek de 5 fouten (10 punten)

De Torens van Hanoi is een puzzel met een aantal schijven. Het spel bestaat uit een plankje met daarop drie pennen (genummerd 1, 2, en 3). Bij aanvang van het spel is op pen 1 een piramidevormige toren van schijven met een gat in het midden geplaatst. Elke schijf heeft een verschillende diameter en de schijven zijn zo geplaatst dat de kleinste bovenop en de grootste onderop ligt. Het doel van het spel is om de complete toren van schijven te verplaatsen naar pen 3, waarbij de volgende regels in acht genomen dienen te worden:

- er mag slechts één schijf tegelijk worden verplaatst
- nooit mag een grotere schijf op een kleinere rusten

Het onderstaande programma lost dit probleem voor drie schijven recursief op. Op de uitvoer vertelt het de gebruiker precies hoe hij/zij de schijven moet verplaatsen. Tevens drukt het programma aan het einde het totaal aantal verplaatsingen af. Voor drie schijven is de correcte uitvoer:

```
Verplaats schijf op pin 1 naar pin 3.
Verplaats schijf op pin 1 naar pin 2.
Verplaats schijf op pin 3 naar pin 2.
Verplaats schijf op pin 1 naar pin 3.
Verplaats schijf op pin 2 naar pin 1.
Verplaats schijf op pin 2 naar pin 3.
Verplaats schijf op pin 1 naar pin 3.
Totaal aantal verplaatsingen: 7
```



Het programma bevat echter 5 fouten. Geef van iedere fout het regelnummer en geef een correctie.

```
 1 #include <stdio.h>
 2
 3 void toonVerplaatsing(int van, int naar) {
 4   printf ("Verplaats schijf op pin %d naar pin %d.\n", van, naar);
 5 }
 6
 7 int hanoi(int aantalSchijven, int van, int naar) {
 8   int aantal;
 9   if (aantalSchijven == 0) {
10     /* basisgeval: er hoeven geen schijven verplaatst te worden */
11   } else {
12     /* recursiegeval */
13     int via = 6 - van - naar;
14     aantal = hanoi(aantalSchijven, van, via);
15     toonVerplaatsing();
16     aantal = aantal + hanoi(aantalSchijven-1, via, naar);
17   }
18   return aantal;
19 }
20
21 int main (int argc, char *argv[]) {
22   aantal = hanoi(3, 1, 3);
23
24   printf("Totaal aantal verplaatsingen: %d\n", aantal);
25   return 0;
26 }
```

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 uitvoert is dus $O(N)$ en niet $O(N^2)$ omdat $O(N)$ de scherpste bovengrens is.

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

2.
```
int i = 0, j = N;
while (j - i > 1) {
  if (i%2 == 0) {
    i = (i + j)/2;
  } else {
    j = (i + j)/2;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.
```
int i = 0;
while (i*i < N) {
  i++;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int i, j = 0, s = 0;
for (i=0; i < N; i++) {
  s += i;
}
for (i=0; i < s; i++) {
  j += 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, s = 0;
for (i=1; i < N; i++) {
  for (j=1; j < i; j*=2) {
    s += j;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6.
```
int i, j, s = 0, a[5] = {0, 0, 0, 0, 0};
for (i=0; i < N; i++) {
  a[i%5]++;
}
for (i=0; i < 5; i++) {
  for (j=0; j < a[i]; j++) {
    s += i + j;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Opgave 4: Eenvoudige algoritmen (20 punten)
(a) Een rij `b` is een *permutatie* van een rij `a` als de rij `b` verkregen kan worden door de elementen van `a` anders te rangschikken. Bijvoorbeeld de rij `b=[1, 2, 4, 3, 5]` is een permutatie van de rij `a=[1, 2, 3, 4, 5]`. Natuurlijk is `b=[1, 2, 4, 3, 5]` niet een permutatie van `a=[1, 2, 3, 4, 1]`, immers de rij `a` bevat geen 5.
Schrijf een routine `isPermutatie` die bepaalt of twee rijen met dezelfde lengte een permutatie van elkaar zijn. Je mag er hierbij van uit gaan dat de rijen alleen cijfers bevatten (dus de waarden 0, 1, .., 9). De functie dient 1 (true) te retourneren als dit het geval is en anders dient de functie 0 (false) te retourneren.
Het prototype van de functie moet zijn: `int isPermutatie(int lengte, int a[], int b[])`
[Puntenwaardering: Je krijgt 10 punten voor een correct algoritme met tijdscomplexiteit O(`lengte`), en 5 punten als het correcte algoritme een slechtere complexiteit heeft.]

(b) De wortel uit 252 is gelijk aan 6 keer de wortel uit 7, want $\sqrt{252} = \sqrt{2^2 \cdot 3^2 \cdot 7} = 2 \cdot 3 \cdot \sqrt{7} = 6\sqrt{7}$. We kunnen $\sqrt{7}$ niet verder vereenvoudigen en daarom noemen we $6\sqrt{7}$ de vereenvoudiging van $\sqrt{252}$. De vereenvoudiging van de wortel uit 504 (=$2 \cdot 252$) is dus $\sqrt{504} = \sqrt{2^3 \cdot 3^2 \cdot 7} = 2 \cdot 3 \cdot \sqrt{2 \cdot 7} = 6\sqrt{14}$.
Schrijf een C-routine `vereenvoudigWortel(int n)` die de vereenvoudiging van `n` op het scherm afdrukt. Je mag er van uit gaan dat `n` niet negatief is. De aanroep `vereenvoudigWortel(252)` dient af te drukken: `wortel(252)=6*wortel(7)`. De aanroep `vereenvoudigWortel(36)` dient af te drukken: `wortel(36)=6`.

Opgave 5: Recursieve algoritmen (20 punten)
(a) In deze opgave beschouwen we het volgende proces. Je begint met een geheel getal a (waarbij $a > 0$) en mag bij herhaling één van de volgende drie operaties uitvoeren:

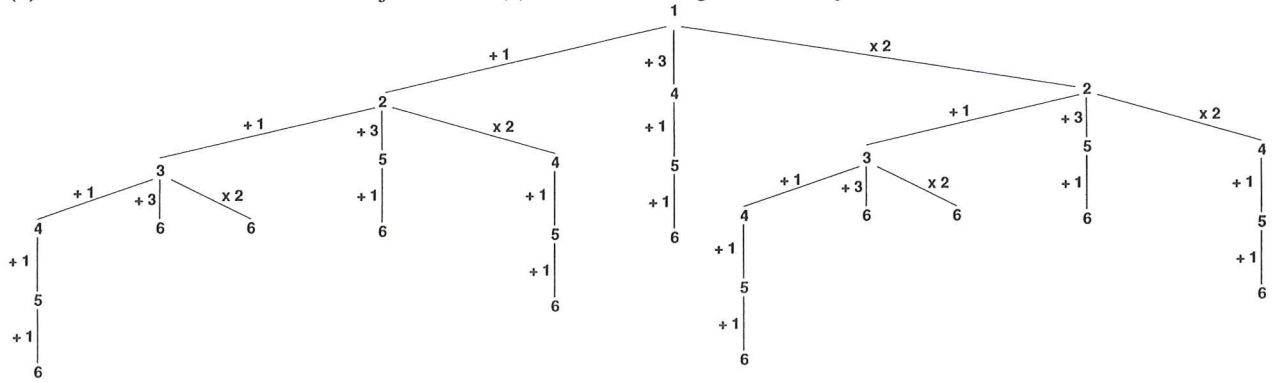
- verhoog a met 1
- verhoog a met 3
- verdubbel a

Met dit proces kunnen we, beginnend met $a = 1$, het getal 6 op 11 manieren bereiken, zoals blijkt uit:

$1+1+1+1+1+1$	$1+1+3+1$	$1\times 2+1+1+1+1$	$1\times 2+3+1$
$1+1+1+3$	$(1+1)\times 2+1+1$	$1\times 2+1+3$	$(1\times 2)\times 2+1+1$
$(1+1+1)\times 2$	$1+3+1+1$	$(1\times 2+1)\times 2$	

Schrijf een recursieve functie met twee parameters `a` en `n`, die het aantal mogelijke manieren oplevert hoe vanuit `a` het getal `n` te bereiken is. De functie mag een exponentiële tijdscomplexiteit hebben (dus een brute-force oplossing is voldoende). Een efficiënte oplosssing wordt gevraagd bij onderdeel (b).

(b) De 11 manieren zoals vermeld bij onderdeel (a) kunnen we ook grafisch weergeven met de volgende boomstructuur.



In deze boom zitten behoorlijk veel gelijke deelbomen: de linker en de rechter deelboom zijn bijvoorbeeld identiek. Maak gebruik van deze eigenschap om met behulp van dynamisch programmeren een efficiënte recursieve oplossing te maken voor het probleem uit onderdeel (a).