Tentamen Imperatief Programmeren

10 november 2011, 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` en `Y` (met hoofdletter!) specificatie-constanten (en dus geen variabelen) zijn.

1.1
```
/* x + 3 == X */
.....
/* x == 2*X + 1 */
```

(a) `x = x/2 + 1;`
(b) `x = 2*x + 5;`
(c) `x = 2*x + 7;`

1.2
```
/* 3*x + 5*y == X */
.....
/* x + 2*y == X */
```

(a) `x = 3*x + 3*y;`
(b) `x = -2*x - 3*y;`
(c) `x = -2*x + 3*y;`

1.3
```
/* y == x*x */
y = y + 2*x + 1; x = x + 1;
.....
```

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

1.4
```
/* x == X, y == Y */
h = y; y = x; x = h;
.....
```

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

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

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

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

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

Opgave 2: Zoek de 5 fouten (10 punten)

Het achtkoninginnenprobleem is een schaakprobleem waarbij acht koninginnen zodanig op een schaakbord (van 8 bij 8) moeten worden geplaatst dat ze elkaar volgens de schaakregels niet aanvallen. Dit betekent dat twee koninginnen niet in dezelfde kolom, rij of diagonaal kunnen staan.

Het onderstaande programma telt het aantal oplossingen van dit probleem met behulp van een recursief algoritme. Het programma bevat echter 5 fouten. Geef van iedere fout het regelnummer en geef een correctie.

```
1  #include <stdio.h>
2
3  int verhoogTeller() {
4    teller++;
5  }
6
7  int abs() {
8    return (a < 0 ? -a : a);
9  }
10
11 void plaatsKoningin(int rij, int pos[8]) {
12   if (rij == 8) {
13     /* basisgeval: we hebben een oplossing gevonden */
14     verhoogTeller();
15   } else {
16     /* recursiegeval */
17     int kolom, r;
18     for (kolom=0; kolom < 8; kolom++) {
19       for (r=0; r <= rij; r++) {
20         if (pos[r] == kolom) { /* koningin op deze kolom? */
21           break;
22         }
23         if (abs(pos[r]-kolom) == rij-r) { /* koningin op diagonaal? */
24           break;
25         }
26       }
27       if (r == rij) { /* veld is niet aangevallen => plaats koningin */
28         pos[rij] = kolom;
29         plaatsKoningin(rij, pos);
30       }
31     }
32   }
33 }
34
35 int main(int argc, char *argv[]) {
36   int pos[8];
37
38   plaatsKoningin();
39   printf("Aantal oplossingen: %d\n", teller);
40
41   return 0;
42 }
```

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

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

3.
```
int i = 0, s = 1;
while (s < N) {
  s += s;
  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, s = 0;
for (i=0; i < N; i++) {
  for (j=0; j < i; j++) {
    s++;
  }
}
```
(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=0; i < N; i++) {
  for (j=i; j >= 0; j/=2) {
    s++;
  }
}
```
(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;
for (i=0; i < N; i += 10) {
  for (j=0; j < 10; 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 *Pythagorees drietal* (a, b, c) bestaat uit drie positieve gehele getallen a, b, c met $a < b < c$ waarvoor geldt $a^2 + b^2 = c^2$. De naam komt van de stelling van Pythagoras, aangezien dergelijke getallen kunnen optreden als de zijden van een rechthoekige driehoek met c als lengte van de schuine zijde. Laat N een positief geheel getal zijn (bijvoorbeeld $N = 1000$). Schrijf een efficiënt programmafragment dat alle Pythagorese drietallen (a, b, c) afdrukt met $a + b + c = N$.
[Puntenwaardering: Je krijgt 10 punten als het algoritme O(N) is, 5 punten als het algoritme O(N*N) is, 1 punt als het minder efficiënt is maar wel correct]

(b) De getallen 567325 en 752653 zijn *getalanagrammen*. Twee getallen x en y zijn getalanagrammen als het getal x volledig bestaat uit de cijfers van y. Hierbij tellen 'leading zeroes' (nullen aan het begin van een getal) niet mee. Dus 001 is niet een anagram van 100, terwijl 1001 wel een anagram is van 1010. Schrijf een efficiënt programmafragment dat van twee integers x en y bepaald of ze wel of niet anagrammen van elkaar zijn.
[Puntenwaardering: Je krijgt 10 punten als het algoritme O(N) is, 5 punten als het algoritme O(N*N) is, waarbij N het aantal cijfers is.]

Opgave 5: Recursieve algoritmen (20 punten)
(a) Schrijf een recursieve functie `keerom` die de omkering afdrukt van zijn integer argument `n`. Dus de aanroep `keerom(12345)` dient `54321` af te drukken. Je hoeft geen rekening te houden met leading zeroes: `keerom(100)` drukt dus `001` af.

(b) Tijdens het college hebben we meerdere malen kennis gemaakt met de rij van Fibonacci. Fibonacci stelde zichzelf de vraag: hoeveel konijnenparen zijn er in jaar n als je uitgaat van de volgende (niet erg realistische) voorwaarden:

- Ieder konijnenpaar bestaat uit één mannetje en één vrouwtje.
- Je begint in jaar 1 met 1 baby-konijnenpaar (dus onvolwassen).
- Elk baby-konijnenpaar is na één jaar volwassen en vruchtbaar.
- Elk volwassen konijnenpaar brengt ieder jaar 1 baby-konijnenpaar voort.
- Geen enkel konijn sterft.

Geef een recursieve functie in C die het aantal konijnenparen aan het begin van jaar n oplevert.

(c) We laten nu de meest onrealistische voorwaarde vervallen, namelijk dat geen enkel konijn sterft. We vervangen deze voorwaarde door "Ieder konijn wordt 5 jaar oud". Geef opnieuw een recursieve functie die het aantal konijnenparen aan het begin van jaar n oplevert onder deze nieuwe voorwaarden.