Tentamen Imperatief Programmeren

8 november 2012, 14:00-17:00

Opgave 1: Toekenningen (20 punten)
De gokkans is 1:3, m.a.w. twee van de zes opgaven wordt statistisch correct gegokt. Er blijven dus 4 'echte' opgaven over. Daarom wordt er per fout antwoord 5 punten afgetrokken.

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

Dus antwoord B.

1.2
```
/* x == 3*X + 1 */
/* (x - 1)/3 == X */
x = (x - 1)/3;
/* x == X */
```

Dus antwoord C.

1.3
```
/* x == y*y + X, y + 1 == Y */
/* x == Y*Y - 2*Y + 1 + X, y + 1 == Y */
y = y + 1;
/* x == Y*Y - 2*Y + 1 + X, y == Y */
/* x + 2*y - 1 == y*y + X, y == Y */
x = x + 2*y - 1;
/* x == y*y + X, y == Y */
```

Dus antwoord B.

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

Dus antwoord C.

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

Dus antwoord C.

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

Dus antwoord B.

Opgave 2: Zoek de 5 fouten (10 punten)

Iedere fout levert twee punten op: één punt voor de gevonden fout, en één punt voor de verbetering van de fout.

```
 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 }
```

- 10: basisgeval retourneert niets. Juist is `aantal = 0;`
- 14: oneindige recursie. Juist is `aantal = hanoi(aantalSchijven-1, van, via);`
- 15. ontbrekende argumenten. Juist is `toonVerplaatsing(van, naar);`
- 16: resultaat is altijd 0. Juist is `aantal = 1 + aantal + hanoi(aantalSchijven-1, via, naar);`
- 22: `aantal` niet gedeclareerd. Juist is `int aantal = hanoi(3, 1, 3);`

Opgave 3: Tijdscomplexiteit (20 punten)
De gokkans is 1:5. Daarom wordt er per fout antwoord 4 punten afgetrokken.

1. ```
   int i = 0, j = N;
   while (i < j) {
     i++;
     j--;
   }
   ```
   Juiste antwoordis C: $O(N)$

2. ```
   int i = 0, j = N;
   while (j - i > 1) {
     if (i%2 == 0) {
       i = (i + j)/2;
     } else {
       j = (i + j)/2;
     }
   }
   ```
   Juiste antwoordis A: $O(\log N)$

3. ```
   int i = 0;
   while (i*i < N) {
     i++;
   }
   ```
   Juiste antwoordis B: $O(\sqrt{N})$

4. ```
   int i, j = 0, s = 0;
   for (i=0; i < N; i++) {
     s += i;
   }
   for (i=0; i < s; i++) {
     j += i;
   }
   ```
   Juiste antwoordis 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;
     }
   }
   ```
   Juiste antwoordis D: $O(N \log N)$

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;
     }
   }
   ```
   Juiste antwoordis C: $O(N)$

Opgave 4: Eenvoudige algoritmen (20 punten)
(a) permutaties zijn vaak handig uit te rekenen m.b.v. een histogram:

```
int isPermutatie(int lengte, int a[], int b[]) {
  int histogram[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  int i;
  for (i=0; i < lengte; i++) {
    histogram[a[i]]++;
    histogram[b[i]]--;
  }
  for (i=0; i<10; i++) {
    if (histogram[i] != 0) {
      return 0;
    }
  }
  return 1;
}
```

(b) Deze opgave is eigenlijk hetzelfde als de factorisatieopgave 5.6.1.7 uit het dictaat:

```
void vereenvoudigWortel(int n) {
  int x = 1;
  int d = 2;
  printf("sqrt(%d)=", n);
  while (d*d <= n) {
    while (n%(d*d) == 0) {
      x = x*d;
      n = n/(d*d);
    }
    d++;
  }
  printf("%d", x);
  if (n > 1) {
    printf("*wortel(%d)", n);
  }
  printf("\n");
}
```

Opgave 5: Recursieve algoritmen (20 punten)
(a) Een brute-force recursieve functie is eenvoudig:

```
int f(int a, int n) {
  if (a == n) return 1;
  if (a > n)  return 0;
  return f(a+1, n) + f(a+3, n) + f(2*a, n);
}
```

(b) Het algoritme wordt veel efficiënter als we reeds berekende deelresultaten opslaan.

```
int fmem(int a, int n, int *mem) {
  if (a == n) return 1;
  if (a > n)  return 0;
  if (mem[a] == 0) {
    mem[a] = fmem(a+1, n, mem) + fmem(a+3, n, mem) + fmem(2*a, n, mem);
  }
  return mem[a];
}

int f(int a, int n) {
  int *mem = calloc(n, sizeof(int));
  int antwoord = fmem(a, n, mem);
  free(mem);
  return antwoord;
}
```