Tentamen Imperatief Programmeren

maandag 2 november 2009, 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.
- 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
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 == X + 2 */
     .....
     /* x == 3*X + 1 */

     (a) x = 3*x - 1;
     (b) x = 3*(X - 1) + 4;
     (c) x = 3*(x - 2) + 1;
```

```
1.2 /* 3*x + 8*y == X */
    .....
    /* 3*x + 5*y == X */

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

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

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

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

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

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

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

```
1.6 /* x == X, y == Y+1 */
    x = x - y; y = x + y; x = y - x;
    .....

    (a) /* x == X + 1, y == X */
    (b) /* x == X, y == Y + 1 */
    (c) /* x == Y + 1, y == X */
```

Opgave 2: Het probleem van de acht koninginnen
Eén van de stukken van het schaakspel is de koningin. Een koningin kan een ander stuk slaan als dit op dezelfde rij, dezelfde kolom of dezelfde diagonaal (twee richtingen) staat. Een schaakbord heeft 64 velden, acht rijen van acht kolommen. In deze opgave beschouwen we het probleem hoe we acht koninginnen zo op een schaakbord kunnen plaatsen dat geen enkele koningin een andere kan slaan.

Omdat op iedere rij van het schaakbord precies één koningin geplaatst dient te worden, kan volstaan worden met de representatie van een bordpositie met een eenvoudig één-dimensionaal array pos (van positie), waarbij pos[rij] de kolom weergeeft van de koningin op de rij met nummer rij .

1

Het onderstaande programma probeert alle 92 oplossingen van dit probleem te vinden en af te drukken op het scherm. Het programma bevat echter 5 fouten. Vind de fouten en geef voor iedere fout een correctie.

```
 1 #include <stdio.h>
 2
 3 void drukAf(int pos[8]) {
 4   int i = 1;
 5   printf ("%d", pos[0]);
 6   while (i < 8) {
 7     printf (" %d", pos[i]);
 8   }
 9   printf ("\n");
10 }
11
12 int abs(int a) {
13   return (a > 0 ? -a : a);
14 }
15
16 void plaatsKoningin(int rij, int pos[8]) {
17   if (rij == 8) {
18     /* basisgeval */
19   } else {
20     /* recursiegeval */
21     for (kolom=0; kolom < 8; kolom++) {
22       int r;
23       for (r=0; r < rij; r++) {
24         if (pos[r] == kolom) { /* koningin op deze kolom? */
25           break;
26         }
27         if (abs(pos[r]-kolom) == rij-r) { /* koningin op diagonaal? */
28           break;
29         }
30       }
31       if (r == rij) { /* veld is niet aangevallen => plaats koningin */
32         pos[rij] = kolom;
33         plaatsKoningin(rij, pos);
34       }
35     }
36   }
37 }
38
39 int main(int argc, char *argv[]) {
40   int pos[8];
41   plaatsKoningin(0, pos);
42   return 0;
43 }
```

Opgave 3: Tijdscomplexiteit
Geef van ieder van de volgende programmafragmenten aan wat de scherpste bovengrens is (in termen van `N`, waarbij $N > 0$) voor het aantal rekenstappen dat het fragment uitvoert. M.a.w. een algoritme dat N stappen doet is $O(N)$ en niet $O(N^2)$ omdat $O(N)$ de scherpste bovengrens is.

3.1
```
int i, som=0;
for (i=N; i >= 0; i--) {
 som += i;
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.2
```
int i=1;
while (i <= N*N) {
 i *= 2;
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.3
```
int i = N;
while (i >= 0) {
 i = (i%2 == 0 ?  i/2 :  i-1);
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.4
```
int i, som=0;
for (i=1; i < N; i++) {
 for (j=N-i; j >= 0; j--) {
  som += j;
 }
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.5
```
int i, j, som=0;
for (i=1; i < N; i*=2) {
 for (j=i+1; j < N; j+=2) {
  som += j;
 }
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3.6
```
int i=0, som=0;
while (som <= N) {
 som += i;
 i++;
}
```

(a) $O(logN)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Opgave 4: Iteratief vs. recursief
De onderstaande functie `macht(int getal, int exponent)` verheft `getal` tot de macht `exponent`.

```
int macht(int getal, int exponent) {
  int resultaat = 1;
  while (exponent > 0) {
    if (exponent % 2 == 1) {
      resultaat *= getal;
      exponent--;
    } else {
      getal = getal*getal;
      exponent /= 2;
    }
  }
  return resultaat;
}
```

(a) De functie maakt gebruik van iteratie. Schrijf een recursieve versie van deze functie.

(b) Een permutatie van een rij tekens is één van de mogelijke manieren om de tekens te rangschikken. Zo heeft het array `rij={'a', 'c', b'}` 6 mogelijke permutaties, namelijk:

- {'a', 'b', 'c'}
- {'a', 'c', 'b'}
- {'b', 'a', 'c'}
- {'b', 'c', 'a'}
- {'c', 'a', 'b'}
- {'c', 'b', 'a'}

Het onderstaande programma drukt alle permutaties van een rij af in lexicografische volgorde, d.w.z. alfabetisch gesorteerd. De functie `int permutaties(int lengte, char rij[])` maakt gebruik van de recursieve functie `recperm`. Implementeer zelf de functie `recperm`.

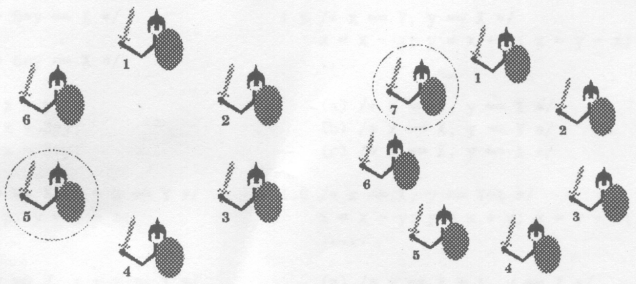
Je mag gebruik maken van alle overige gegeven functies.

```
#include <stdio.h>

void verwissel(int i, int j, char rij[]) {
  char h = rij[i];
  rij[i] = rij[j];
  rij[j] = h;
}

void sorteer(int lengte, char rij[]) {
  int i, j;
  for (i=0; i < lengte; i++) {
    for (j=i+1; j < lengte; j++) {
      if (rij[j] < rij[i]) {
        verwissel(i, j, rij);
      }
    }
  }
}

void drukAf(int lengte, char rij[]) {
```

```
  int i;
  printf("%c", rij[0]);
  for (i=1; i < lengte; i++) {
    printf(" %c", rij[i]);
  }
  printf("\n");
}

void recperm(int idx, int lengte, char rij[]) {
 ..... /* implementeer deze functie zelf */
}

int permutaties(int lengte, char rij[]) {
  recperm(0, lengte, rij);
}

int main(int argc, char *argv[]) {
  char rij[] = {'b', 'c', 'a'};
  permutaties(3, rij);
  return 0;
}
```

Opgave 5: Josephus probleem

Een klassiek probleem uit de informatica is het zogenaamde Josephus probleem. Gegeven n personen, genummerd 1 tot en met n, die in een cirkel staan en een geheel getal s (met $1 \leq s \leq n$). Elimineer, te beginnen bij persoon s, telkens elke tweede persoon, totdat er nog maar één persoon over is. Het nummer van de overgebleven persoon noemen we $J(n, s)$.



Figuur 1: (links) J(6,2)=5, (rechts) J(7,2)=7.

(a) Schrijf een functie die voor gegeven n en s de waarde $J(n, s)$ bepaalt.

(b) Schrijf een functie die voor een gegeven n een s bepaalt zodanig dat $J(n, s) = 1$.