Uitwerkingen Tentamen Inleiding Programmeermethoden
vrijdag 19 maart 2004

OPGAVE 1

a.
```
int kleinste (double A[ ], int i, int n) {
     int kl = i, k;
     for ( k = i+1; k < n; k++ )
       if ( A[k] < A[kl] ) kl = k;
     return kl;
   }//kleinste
```

b.
```
void schuif (double A[ ], int i, int j) {
     int k; double temp = A[j];
     for ( k = j-1; k >= i; k-- ) A[k+1] = A[k];
     A[i] = temp;
   }//schuif
```

c.
```
void sorteer (double A[ ], int n) {
     int i, j;
     for ( i = 0; i < n-1; i++ ) {
       j = kleinste (A, i, n);
       if ( i < j ) schuif (A, i, j);
     }//for
   }//sorteer
```

d. Een keer schuif (A,i,n-1) kost n-i+1 toekenningen van double's
(steeds geldt namelijk dat j = n-1, de kleinste staat telkens achteraan).
Dus: n+1 + n + ... + 4 + 3 = (n-1)(n+4)/2, kortom orde n-kwadraat

e. Een maal een echte schuif; met k = 2j+1: i-j+1 = i-k/2+3/2 stuks

OPGAVE 2

a. Globale variabelen gelden in het gehele programma, en worden helemaal bovenin aangemaakt. Locale variabelen gelden (tijdelijk) alleen in de functie waarin ze aangemaakt zijn.
Variabelen kunnen call by value en call by reference worden meegegeven aan een functie. Bij call by value gaat alleen de waarde van de parameter naar de functie, alwaar een locale variabele deze waarde opvangt, en er met deze locale variabele wordt verder gerekend. De oorspronkelijk variabele behoudt zijn waarde. Bij call by reference (&) gaat als het ware de variabele zelf naar de functie, en kan dan ook blijvend veranderd worden. Eigenlijk wordt het adres (de reference) doorgegeven.

b. 3 5 4 8 20 20 5.0 1 4 8
OF: 3 8 2 4 17 17 4.25 1 4 8 als + van rechts naar links "gaat"

c. 4 5 3 8 19 21 5.0 2 4 7 (en hier: 3 8 4 5, rest hetzelfde)

d. 6 6 5 5 22 22 5.5 4

e. gem (a, a, a, a) levert a-1.5 op

f. Het mag, mits eerste en tweede parameter van gem call by value zijn, dus zonder &. Het is geen recursie (tenzij het statement in de functie gem zelf staat).

OPGAVE 3

a.
```
int groter (int Life[ ][n], int X, int m) {
     int tel = 0; int i, j;
     for ( i = 0; i < m; i++ )
       for ( j = 0; j < n; j++ )
         if ( Life[i][j] > X ) tel++;
     return tel;
   }//groter
```

b.
```
bool max (int Life[ ][n], int i, int j, int m) {
     return ( groter ( Life, Life[i][j]-1, m) == 1 );
   }//max
```

c.
```
void overleef (int Life[ ][n], const int m) {
     int i, j, tel; int Lifekopie[m][n];
     for ( i = 0; i < m; i++ )
       for ( j = 0; j < n; j++ ) {
         tel = 0;
```

```
        if ( i > 0 )   tel += Life[i-1][j];
        if ( i < m-1 ) tel += Life[i+1][j];
        if ( j > 0 )   tel += Life[i][j-1];
        if ( j < n-1 ) tel += Life[i][j+1];
        if ( 4 <= tel && tel <= 8 ) Lifekopie[i][j] = Life[i][j];
        else Lifekopie[i][j] = 0;
      }//for
    for ( i = 0; i < m; i++ ) for ( j = 0; j < n; j++ )
      Life[i][j] = Lifekopie[i][j];
  }//overleef
d. int hoeveel (int Life[ ][n], int i, int j, int m) {
    bool naarrechts = true; int som = 0; int geval = 0;
    while ( geval < 2 ) {
      if ( i == m || j == n || Life[i][j] == 0 ) {
        if ( naarrechts ) j--; else i--;
        geval++; naarrechts = ! naarrechts; }//if
      else { som += Life[i][j]; geval = 0; }//else
      if ( naarrechts ) j++; else i++;
    }//while
    return som;
  }//hoeveel

OPGAVE 4
a. void toevoegen (dag* & serie, int numero) {
    dag* nieuw = new dag; nieuw->nummer = numero;
    nieuw->volgende = serie; nieuw->eenander = NULL; serie = nieuw;
  }//toevoegen
b. void verwijder (dag* & serie) {
    dag* weg; dag* hulp = serie;
    serie = serie->volgende;  // gegeven: niet NULL
    weg = serie->eenander; serie->eenander = hulp;
    if ( weg != NULL ) delete weg;
  }//verwijder
c. void wissel (dag* serie) {
    int temp;
    if ( serie != NULL && serie->volgende != NULL )
      if ( serie->nummer > serie->volgende->nummer ) {
        temp = serie->nummer;
        serie->nummer = serie->volgende->nummer;
        serie->volgende->nummer = temp;
      }//if
  }//wissel
d. Bij a en b moet er een & bij: de ingangspointer zal (bij b meestal)
  gaan veranderen.
  Bij c maakt het niet uit, de ingangspointer verandert toch niet.
e. int buiten (dag* serie) {
    int tel = 0; bool inlijst; dag* p = serie; dag* q;
    while ( p != NULL ) {
      if ( p->eenander != NULL ) {
        q = serie; inlijst = false;
        while ( q != NULL ) {
          if ( p->eenander == q ) inlijst = true;
          q = q->volgende;
        }//while
        if ( ! inlijst ) tel++;
      }//if
      p = p->volgende;
    }//while
    return tel;
  }//buiten
```