Tentamen Inleiding programmeren voor LS&Ters
Vrijdag 12 maart 2004, 14.00–17.00 uur
Universiteit Leiden — Informatica

Bij de te schrijven functies moeten de variabelen in de heading voorkomen of locaal zijn (niet stiekem globale variabelen gebruiken). De opgaven tellen alle vier even zwaar mee. Veel succes! Cijfers te zijner tijd: `http://www.liacs.nl/home/kosters/lst/`

1. **a.** Schrijf een C++-functie `double min (double x, double y, double z)` die het kleinste van de drie getallen `x`, `y` en `z` met behulp van een `return`-statement teruggeeft.
b. Schrijf een C++-functie `int som (int n)` die met behulp van een `for`-loop de volgende sommatie uitrekent: $1^2 + 2^2 + \ldots + n^2$.
c. Schrijf een C++-functie `void deling (int x, int y)` die zowel het aantal keer dat `x` door `y` gedeeld kan worden op het beeldscherm afdrukt, als de rest bij deling van `x` door `y`. Doe dit door herhaald `y` van `x` af te trekken. Er mag dus geen gebruik van `/` en/of `%` gemaakt worden. Neem aan dat de getallen `x` en `y` beide positief (> 0) zijn. Als voorbeeld: voor 26 en 4 moeten 6 en 2 worden afgedrukt.
d. Gegeven zijn twee arrays `A` en `B`, beide met 10 karakters. Schrijf een C++-functie `bool spiegel (char A[10], char B[10])` die nagaat of `A` hetzelfde is als `B`, maar dan van achter naar voren (de functie moet in dat geval `true` opleveren, en anders `false`). Hierbij moeten de karakters exact gelijk zijn, terwijl een `'*'` ieder karakter "matcht". Een voorbeeld: de twee array's `'a' '1' '*' 'm' 'p' 'T' '*' '4' '4' 'Q'` en `'Q' '4' '4' '*' 'T' '*' 'm' '4' '1' 'a'` leveren `true`. Gebruik een `while`-loop.

2. Gegeven een array `A` (`double A[n];`, met `const n = 1000;`) met `n` verschillende getallen.
a. Schrijf een C++-functie `int kleinste (double A[n], int i)` die de *index* oplevert van de kleinste waarde uit `A[i]` tot en met `A[n-1]`.
b. Geef een C++-functie `void schuif (double A[n], int i, int j)` die de elementen `A[i]` tot en met `A[j-1]` alle één plaats naar rechts opschuift en `A[j]` op positie `i` zet. Neem aan dat $i < j < n$. Voorbeeld: `A` is 1.0 2.0 <u>6.0 8.0 5.0 3.0</u> 4.0; $i = 2$; $j = 5$; dan is het gevolg van `schuif`: 1.0 2.0 <u>3.0 6.0 8.0 5.0</u> 4.0 .
c. Schrijf nu een C++-functie `void sorteer (double A[n])` die het array oplopend sorteert door herhaald `kleinste` en `schuif` aan te roepen. Doe dit net als bij "simpelsort": het beginstuk is steeds oplopend gesorteerd, zoek herhaald de kleinste van de (steeds kleiner wordende) rest en schuif die naar het begin van de rest.
d. Stel dat je deze sorteermethode toepast op het rijtje `n n-1 ... 3 2 1`. Hoeveel "verschuivingen" (toekenningen van `double`'s) doet het algoritme van **c** dan in totaal?

3. **a.** Bij een functie kun je te maken hebben met *call by value* en *call by reference*, en ook met *locale* en *globale* variabelen. Verder heb je ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.
b. Een zeker C++-programma bevat de volgende programmaregels:

```
int desom (int & een, int & twee) {
  een--; twee++; cout << een << twee << endl;
  return een + twee;
}//desom
```

```
double gem (int & a, int & b, int & c, int & d) {  // gemiddelde
  int deel1 = desom (a, b), deel2 = desom (c, d);
  return desom (deel1, deel2) / 4.0;
}//gem
```

Gegeven de volgende programma-code:

```
a = 5; c = 4; h = 7; t = 1;
cout << gem (a, c, h, t); cout << a << c << h << t << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg. De variabelen `a`, `c`, `h` en `t` zijn globaal en van type `int`.
c. Als **b**, maar nu met alle zes `&`'s weggelaten.
d. Weer terug naar de situatie van **b**, met de zes `&`'s erbij. Wat gebeurt er bij uitvoering van het volgende stukje C++? Wat wordt er afgedrukt?

```
a = 7; cout << gem (a, a, a, a) << endl; cout << a << endl;
```

En wat levert `gem (a, a, a, a)` in het algemeen op — als functie van `a`?
e. Wat zou de reden zijn dat in `gem` door `4.0` gedeeld wordt en niet gewoon door `4`?
f. Mag een statement als `a = (int) gem (desom (a, c), desom (c, h), h , t);` ergens in het programma staan? En zo ja, is er dan sprake van recursie?

4. We hebben een 2-dimensionaal array `Life` met 100 rijen en 100 kolommen, gevuld met positieve gehele getallen: `int Life[100][100];`. In het voorbeeld hieronder staat het gedeelte "linksboven" van zo'n array:

```
3  5  7 ...
1  2  5 ...
7  7 11 ...
...
```

Zo geeft het getal `3` aan dat er op positie `(0,0)` drie levende cellen zitten.
a. Geef een C++-functie `int groter (int Life[100][100], int X)` die oplevert hoeveel getallen uit `Life` echt groter zijn dan `X`.
b. Schrijf een C++-functie `bool max (int Life[100][100], int i, int j)` die `true` oplevert als `Life[i][j]` het grootste array-element is, en anders `false`. Als het getal meerdere keren voorkomt, en dus niet het enige maximum kan zijn, moet de functie ook `false` geven. Hierbij *moet* de functie van **a** (verstandig) gebruikt worden.
c. Schrijf een C++-functie `void overleef (int Life[100][100])` die alleen die cellen laat overleven die in vakjes zitten waarvoor in totaal tussen 4 en 8 cellen in de directe buren aanwezig zijn. Een en ander gebeurt voor alle array-elementen tegelijk (parallel). Tel voor alle array-elementen de aantallen cellen uit de vier (aan de randen van het array twee of drie) horizontale en verticale buren op, en als dit aantal kleiner dan 4 of groter dan 8 is maak je te zijner tijd het oorspronkelijke array-element 0; anders blijft het onveranderd. In het voorbeeld-array blijft de 3 bestaan, de 5-en wordt 0-en, etcetera.
d. Welke van de twee `100`'s in de kopregels van de functies hierboven mag wel/niet weggelaten worden, en waarom?