Tentamen Programmeermethoden
Maandag 2 april 2007, 14.00–17.00 uur
Universiteit Leiden — Informatica

Bij alle te schrijven functies moeten de variabelen (constanten uitgezonderd) in de heading of als locale variabele voorkomen; vul zelf de headings goed in. De opgaven tellen alle vier even zwaar mee. Succes! Cijfers: `http://www.liacs.nl/home/kosters/pm/res06.txt`.

1. In deze opgave zijn alle gehele getallen > 0. In een array `int A[n]` stoppen we `n` (een `const` > 0) gehele getallen.
a. Schrijf een C++-functie `bubblesort (A,n)` die `A` oplopend sorteert met *bubblesort*.
b. Schrijf een C++-functie `bool saai (x)` die `true` teruggeeft precies als het positieve gehele getal `x` uit allemaal dezelfde cijfers bestaat. Zo'n getal noemen we *saai*. Voorbeelden: 3333 en 22. Maar niet: 16, 166 en 555255.
c. We willen in het array `A` alle saaie getallen voor alle andere getallen zetten. Schrijf een C++-functie `saaivooraan (A,n)` die dit doet, en wel als volgt. Gebruik tellers `i` en `j`, die vooraan respectievelijk achteraan beginnen. Als `A[j]` saai is en `A[i]` niet: wissel ze; anders: laat `i` en `j` handig enigszins naar elkaar toegaan.
d. Neem aan dat de saaie getallen reeds voor de andere getallen staan. Zorg er nu ook voor dat de saaie getallen oplopend gesorteerd zijn, en evenzo de andere getallen, maar dat de saaie vooraan blijven. Schrijf hiertoe een aangepaste `bubblesort2 (A,n)`, met zo min mogelijk aanvullingen op de versie van **a**: alleen de test mag aangepast worden.

2.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 onderscheiden we ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.
b. Gegeven een C++-programma met daarin de volgende twee functies:

```
int wiebenik (int a, int b) {
  int som = 0, t = a*a;
  while ( a <= b ) { som += t; t = t + 2*a + 1; a++;
    cout << a << ", " << t << " en " << som << endl; }//while
  return som;
}//wiebenik
void tabel (int a, int b) {
  for ( a = a; a <= b; a++ ) { som += wiebenik (a,b);
    cout << a << ", " << b << " en " << som << endl; }//for
}//tabel
```

Verder zijn de globale variabelen `x`, `y` en `som` gegeven (alle van type `int`). Voordat de functie `tabel` wordt aangeroepen hebben zij de waarde 2, 4 en 0 respectievelijk. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
tabel (x,y); cout << x << ", " << y << " en " << som << endl;
```

c. Als **b**, maar nu staat er een `&` bij alle vier parameters.
d. Als **b**, maar nu met alleen twee `&`'s in `wiebenik`, en

```
tabel (x,x); cout << x << ", " << y << " en " << som << endl;
```

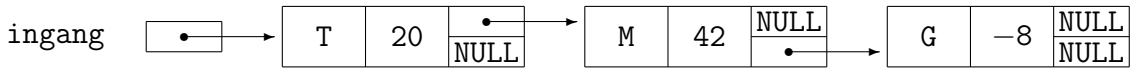
e. Wat berekent (in woorden) de functie `wiebenik (a,b)`, uitgedrukt in `a` en `b`?

3. Gegeven is een `n` bij `n` (met `n` een `const` ≥ 2) array `kost`: `int kost[n][n];`, gevuld met gehele getallen ≥ 0. Een getal `kost[i][j]` > 0 stelt de kosten voor om rechtstreeks van `i` naar `j` te reizen ($0 \leq$ `i, j` $<$ `n` en `i` $\neq$ `j`), en `kost[i][j]` $= 0$ betekent dat er geen directe verbinding is van `i` naar `j`. Er geldt `kost[i][i]` $= 0$ voor $0 \leq$ `i` $<$ `n`; verder is vanuit iedere plaats minstens één directe reis mogelijk. Als je rechtstreeks van `i` naar `j` kunt reizen, kun je ook direct van `j` naar `i`; maar misschien verschillen de kosten wel!
a. Schrijf een C++-functie `double gem (kost,i)` die bepaalt wat de gemiddelde kosten zijn voor een directe reis vanuit `i` ($0 \leq$ `i` $<$ `n`). Denk eraan te middelen over de "niet-nullen".
b. Schrijf een functie `verschil (kost)` die het grootste absolute verschil uitrekent dat je kunt hebben tussen directe reizen van `i` naar `j` en `j` naar `i` (tussen heen- en terugreizen dus), voor willekeurige `i` en `j`. Onderzoek hiertoe alle paren (`i, j`).
c. Schrijf een C++-functie `hoeveel (kost,i)` die bepaalt hoeveel plaatsen je vanuit een zekere `i` ($0 \leq$ `i` $<$ `n`) kunt bereiken, inclusief `i` zelf, waarbij je zo vaak je wilt mag "overstappen". Hint: gebruik een array `bool D[n]`, initieel gevuld met `false`'s (alleen `D[i]` $=$ `true`), waar in `D[j]` moet komen of je `j` kunt bereiken.

4. Gegeven is het volgende type:

```
class mens { public: mens* volg1; mens* volg2; char naam; int leeftijd; };
```

Met behulp hiervan worden rijtjes (lijstjes) met mensen opgebouwd; alle gebruikte namen en leeftijden ($\neq 0$) verschillen onderling. *Of* het veld `volg1` *of* het veld `volg2` bevat een pointer naar het volgende `mens`-object (of zijn beide `NULL`). Bij vrouwen gebruiken we `volg1`, bij mannen `volg2`. Van de laatste mens in de lijst slaan we als het een man betreft de leeftijd negatief op. Een voorbeeld (`ingang` van type `mens*`):



We hebben dus een 20-jarige vrouw `T`, een 42-jarige man `M` en een 8-jarige man `G`.
a. Schrijf een C++-functie `voegtoe (ingang,nm,lt,mv)` die een nieuw mens, geheten `nm`, `lt` jaar oud, vooraan de structuur (met `ingang` van type `mens*` als ingang) toevoegt. De `bool mv` bepaalt of het een man (`false`) of vrouw (`true`) is. Denk aan de lege lijst!
b. Schrijf een C++-functie `verwijderman (ingang)` die het eerste object uit de lijst (met `ingang` van type `mens*` als ingang) verwijdert indien daarin een man zit. Denk aan de (bijna) lege lijst.
c. Schrijf een C++-functie `verwissel (ingang)` die de *leeftijden* van eerste en tweede mens netjes verwisselt — mits ze bestaan natuurlijk. Denk aan het speciale geval dat de tweede en laatste mens een man is!
d. In de functies bij **a**, **b** en **c** staat in de heading de parameter `ingang`. Deze heb je call by value of call by reference doorgegeven (met een `&`). Maakt het voor de werking van deze functies verschil uit of die `&` erbij staat? Leg duidelijk uit.
e. Schrijf een functie `int vrouwman (ingang)` die telt hoevaak het voorkomt dat direct na een vrouw een man komt in de lijst.