Tentamen Programmeermethoden
Maandag 31 juli 2006, 10.00–13.00 uur
Universiteit Leiden — Informatica

Bij alle te schrijven functies moeten de variabelen in de heading of als locale variabele voorkomen; vul zelf de headings goed in. De opgaven tellen alle vier even zwaar mee. Veel succes! Cijfers: `http://www.liacs.nl/home/kosters/pm/res05.txt`.

1. a. In een array `A` (`int A[n]`, met constante `n`) slaan we de maximaal `n` cijfers van een positief geheel getal (> 0) op. Het laatste cijfer zit in `A[n-1]`. Bijvoorbeeld, met `n` $= 100$: voor het getal 206 hebben we `A[99]=6`, `A[98]=0`, `A[97]=2` en de overige `A[i]`'s zijn `0`. Schrijf een Boolese functie `iseen (A,n)` die precies dan `true` oplevert als `A` het getal 1 voorstelt.
b. Schrijf een functie `kopie (A,B,n)` die de inhoud van `B` in `A` kopieert.
c. Schrijf een Boolese functie `telop (A,B,n)` die de "getallen" `A` en `B` in `A` optelt (dus `A` wordt "`A` plus `B`"). Als het "past" moet er `true` worden teruggeven, bij overflow (`A[0]` te groot) `false`. In dat laatste geval doet de inhoud van `A` aan het eind er niet toe.
d. Schrijf een functie `min1 (A,n)` die `1` aftrekt van `A`. Aanname: getal was groter dan 0.
e. Schrijf een Boolese functie `maal (A,B,n)` die de "getallen" `A` en `B` in `A` vermenigvuldigt (dus `A` wordt "`A` keer `B`"). Wederom: retourneer `false` bij overflow. Doe dit door herhaald `A` bij zichzelf op te tellen. Gebruik **a**, **b**, **c** en **d**. Neem aan dat `B` een getal > 0 voorstelt.

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 bend (int x, int y, int z) {
   int hulp = x; x = y; y = z; z = hulp;
   cout << x << "," << y << "," << z << endl;
   return x+y+z;
}//bend
int snap (int a, int b, int c) {
   int i, som = 0;
   for ( i = 1; i <= c; i++ ) { a--; som += bend (a,b,c); }//for
   cout << a << "," << b << "," << c << endl;
   return som;
}//snap
```

Verder zijn de globale variabelen `x`, `y` en `z` gegeven (alle van type `int`). Voordat de functie `snap` wordt aangeroepen hebben zij de waarde 5, 6 en 4 respectievelijk. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
cout << snap (x,y,z) << endl; cout << x << "," << y << "," << z << endl;
```

c. Als **b**, maar nu staat er een `&` bij alle zes parameters.
d. Als **c** (dus weer met zes `&`'s), maar nu met `cout << snap (x,x,x) << endl;`.
e. Je wilt nu binnen de functie `bend` de functie `snap` aanroepen. Dit is dan (indirecte) recursie. Wat moet je dan nog toevoegen in je programma en waarop moet je letten als het gaat om de werking van het programma?

3. We hebben een array `int S[n][n]` (met constante `n`) met gehele getallen ≥ 0. Dit stelt hier een geheel of gedeeltelijk ingevulde *Sudoku* voor. Lege vakjes geven we aan met `0`, de overige vakjes bevatten een getal tussen `1` en `n`. We bekijken de eenvoudige variant van Sudoku waarbij alleen in rijen en kolommen elk getal uit `1` tot en met `n` precies één keer moet voorkomen. (De gebruikelijke restricties voor blokken vervallen.)
a. Schrijf een functie `int leeg (S)` die het aantal lege vakjes in S bepaalt.
b. Schrijf een functie `bool okee (S,i)` die controleert of elke positieve waarde `1` tot en met `n` precies één keer in rij `i` voorkomt. Gebruik hierbij eventueel een hulparray `A`.
c. Schrijf een functie `int invulbaar (S,i,j,k)` die voor een gegeven leeg vakje (`i`, `j`) in een tot dusver goed ingevulde Sudoku berekent hoeveel getallen daar volgens de regels kunnen worden ingevuld, en de kleinste van die mogelijkheden teruggeeft in de parameter `k`. Als er 0 mogelijkheden zijn, maak `k` dan ook `0`.
d. We gaan nu proberen de oorspronkelijke Sudoku verder in te vullen. Schrijf hiertoe een functie `bool invullen (S)` die als volgt te werk gaat: zoek een vakje (maakt niet uit welk) waarvoor het aantal in te vullen waardes $\neq 0$ is. Gebruik hier **c**; vul de in `invulbaar` opgeleverde `k`-waarde in. Herhaal dit totdat er geen lege vakjes meer zijn (de Sudoku is opgelost; retourneer `true`) of totdat er geen invulbare vakjes meer zijn (geef `false`).

4. Gegeven is het volgende type:

```
class mens { public:
  mens* volgende;
  int aantal; info leeftijd; };//mens
```

Met behulp hiervan worden rijtjes (lijstjes) mensen opgebouwd. Het veld `volgende` bevat steeds een pointer naar het direct "rechts" ernaast gelegen vakje. In `leeftijd` staat de leeftijd van de persoon, `aantal` is het aantal vakjes met dezelfde leeftijd die er nog nakomen. Een voorbeeld (`ingang` van type `mens*`):

```
ingang ---> 2 18 ---> 0 22 ---> 1 18 ---> 0 18 ---> 0 33 NULL
```

De `2` in het eerste vakje betekent dat er in totaal 3 mensen 18 zijn.
a. Schrijf een C++-functie `verwijder (ingang)` die het eerste `mens`-vakje uit de lijst (met `ingang` van type `mens*` als ingang) verwijdert, mits dat vakje bestaat en die persoon minstens 18 is.
b. Schrijf een C++-functie `voegtoe (ingang,lt)` die een nieuw vakje vooraan de structuur (met `ingang` van type `mens*` als ingang) toevoegt, met leeftijd `lt`. Er mag gebruik gemaakt worden van een functie die het aantal mensen van een bepaalde leeftijd teruggeeft, zie **e**.
c. Schrijf een C++-functie `wissel (ingang)` die de leeftijden van de eerste twee vakjes (mensen) verwisselt, indien er minstens twee zijn — en de `aantal`-velden aanpast.
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 ook uit wat er bij deze twee mogelijkheden precies gebeurt tijdens executie van de betreffende functie.
e. Schrijf een efficiënte C++-functie `int hoeveel (ingang,lt)` die teruggeeft hoeveel mensen in de lijst leeftijd `lt` hebben.