Tentamen Programmeermethoden
Donderdag 17 december 2009, 10.00–13.00 uur
Universiteit Leiden — Informatica

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

1. In een array `int A[n]` staan `n` (een *oneven* `const` > 0) verschillende gehele getallen.
a. Schrijf een C++-functie `double gem (A,n)` die het gemiddelde van de getallen uit `A` teruggeeft. Zorg er voor dat dit gemiddelde als een `double` wordt uitgerekend.
b. Schrijf een C++-functie `bool gr (X,t,A,n)` die bepaalt of er *precies* `t` getallen in `A` groter dan de integer `X` zijn, voor gegeven $0 \leq$ `t` $\leq$ `n`. Stop zodra het antwoord bekend is.
c. Schrijf een C++-functie `mediaan (A,n)` die de *mediaan*, oftewel het middelste getal in grootte, van `A` oplevert. Gebruik **b**. Voor de mediaan geldt dat er in `A` precies evenveel getallen groter als kleiner zijn. Het gaat hier om de array-waarde, niet om de array-index.
d. Schrijf een C++-functie `sort (A,n)` die `A` oplopend sorteert met behulp van *bubblesort*.
e. Als `A` oplopend gesorteerd wordt met **d**, levert `return A[n/2];` ook de mediaan op. Bespreek de efficiëntie van deze methode in relatie tot die van **c**.

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 somofprod (bool jn, int  x, int y) {
  jn = !jn;
  if ( jn ) { cout << "Som" << endl; x--; return (x+1) + y; }//if
  else { cout << "Product" << endl; return x * y; }//else
}//somofprod

int detest (int a, int b) {
  int i; bool jn;
  for ( i = 1; i <= a; i++ ) {
    jn = ( i < b ); b += somofprod (jn,a,b);
    cout << a << " en " << b << endl;
  }//for
  return a + b;
}//detest
```

Verder zijn de globale variabelen `x`, `y` en `z` gegeven (alle van type `int`). Voordat de functie `detest` wordt aangeroepen hebben zij de waarde `1`, `3` en `0`, respectievelijk. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x = detest (y,z); cout << x << " ," << y << " en " << z << endl;
```

c. Als **b**, maar nu met een `&` (“ampersand”) bij de vijf parameters van de functies.
d. Als in de functie `detest` staat `b += somofprod (( i < b ),a,b);`, compileert het programma dan nog? Onderscheid de gevallen met en zonder `&`, zie **b** en **c**.
e. Vervang `b += somofprod (jn,a,b);` door `a += somofprod (jn,a,b);`. Wat gaat er mis in de werking van het programma? Of de `&`'s erbij staan maakt overigens niet uit.

3. Gegeven is een `m` bij `n` (beide `const` > 0) array `bebouwing`, gevuld met getallen ≥ 0; `bebouwing[i][j]` stelt de bebouwing op punt $(\texttt{i}, \texttt{j})$ voor, waarbij een 0 staat voor een weg.

```
42  0  9 5 18  0
 0  0  0 0 18 12
 0 17 14 0  8 17
```

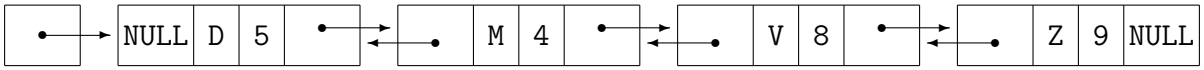
a. Schrijf een C++-functie `uitbreiden (bebouwing,i,j,X)` die —mits punt $(\texttt{i}, \texttt{j})$ met `X` bebouwd is— diens (maximaal) vier directe buren (horizontaal of verticaal) ook met `X` bebouwt, als daar een weg is. Bij aanroep `uitbreiden (bebouwing,2,2,14)` zouden in het voorbeeld twee wegen met `14` bebouwd worden, namelijk de punten $(1, 2)$ en $(2, 3)$. En `uitbreiden (bebouwing,2,2,17)` doet niets.
b. Schrijf een C++-functie `bool bereikbaar (bebouwing,i,j,p,q)` die precies dan `true` oplevert als de twee bebouwde punten $(\texttt{i}, \texttt{j})$ en $(\texttt{p}, \texttt{q})$ vanuit elkaar te bereiken zijn door herhaald horizontaal of verticaal over aangrenzende wegen te lopen. Voorbeeld: vanuit $(0, 0)$ is $(2, 4)$ bereikbaar, maar $(1, 5)$ niet.
Tip: zet een -1 in `bebouwing[i][j]`; breid vervolgens *vaak genoeg* uit met behulp van **a**. Grenst $(\texttt{p}, \texttt{q})$ nu aan een -1? Let er wel op `bebouwing` na afloop weer op de oorspronkelijke waardes terug te zetten. Gebruik geen recursie.
c. Schrijf een C++-functie `bool goedzo (bebouwing)` die precies dan `true` oplevert als alle bebouwde punten vanuit elkaar via de weg bereikbaar zijn, als beschreven in **b**. In het voorbeeld: `false`.

4. Gegeven is het volgende type:

```
class dag { public: char naam; int datum; dag* vorig; dag* volg; };
```

Met behulp hiervan worden lijstjes met namen van dagen (hoofdletters) en hun datum opgebouwd. Het veld `volg` bevat een pointer naar het volgende `dag`-object, en de `vorige`-pointer wijst naar het vorige `dag`-object. Een voorbeeld (`jan2010` van type `dag*`):

`jan2010`



a. Schrijf een C++-functie `verwissel (jan2010)` die de `naam`- en `datum`-inhouden van de twee eerste objecten — indien aanwezig — in de juiste tijdsvolgorde (gelet op de datum) zet. In het voorbeeld moeten de `D` en de `M` verwisseld worden, evenals de `5` en de `4`. Je mag ook de objecten via de pointers verwisselen, als je dat liever doet.
b. Schrijf een C++-functie `voegtoe (jan2010,nm,dt)` die een nieuwe dag met naam `nm` en datum `dt` erin vooraan de lijst (met `jan2010` van type `dag*` als ingang) toevoegt.
c. Schrijf een C++-functie `verwijder (jan2010)` die de *tweede* dag uit de lijst (met `jan2010` van type `dag*` als ingang) verwijdert, mits die er is. Denk dus aan de lege lijst en aan een lijst met één element.
d. In de functies bij **a**, **b** en **c** staat in de heading een pointer. 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 C++-functie `dag* kleiner (jan2010)` die een pointer naar de eerste dag oplevert (eventueel `NULL`) die een datum kleiner dan zijn directe voorganger heeft. In ons oorspronkelijke voorbeeld: een pointer naar de tweede dag uit de lijst.