Tentamen Programmeermethoden
Maandag 10 januari 2005, 14.00–17.00 uur
Universiteit Leiden — Informatica

Bij alle te schrijven functies moeten de variabelen in de heading of als locale variabele voorkomen (niet stiekem globale variabelen gebruiken). De opgaven tellen alle vier even zwaar mee. Veel succes! Cijfers: `http://www.liacs.nl/home/kosters/pm/res04.txt`.

1. Gegeven `const int n = 1000;`, en een array `A` met `n` gehele getallen (`int A[n]`), die niet verschillend hoeven te zijn. Geef de functies zelf de juiste headings (met type e.d.).
a. Schrijf een C++-functie `void bubblesort (A,n)` die het array `A` oplopend sorteert met behulp van *bubblesort*. Gelijke waarden komen achter elkaar in het array te staan.
Gegeven is vanaf nu dat `A` oplopend gesorteerd is.
b. Schrijf een C++-functie `int verschillend (A,n)` die het aantal verschillende getallen uit `A` oplevert. Eén keer door `A` lopen is voldoende. Voorbeeld: het rijtje 2 4 4 4 6 7 7 7 7 7 7 9 bevat 5 verschillende getallen.
c. Gegeven is een index `ind` en een geheel getal `waarde`. Schrijf nu een C++-functie `int hoevaak (A,n,ind,waarde)` die bepaalt hoe vaak `waarde` in `A[ind]` tot en met `A[n-1]` voorkomt. Hierbij dient gebruikt te worden dat `A` oplopend gesorteerd is.
d. Schrijf een C++-functie `void vaakst (A,n,meest,zovaak)` die de waarde `meest` bepaalt die het vaakst in `A` voorkomt. De waarde `zovaak` geeft dan aan hoe vaak `meest` voorkomt. Voor het voorbeeld uit **b** zou de functie een waarde `meest` is 7 en `zovaak` is 6 opleveren. Je *kunt* hierbij de functie uit **c** handig gebruiken. Als er twee of meer waarden zijn die het vaakst voorkomen, laat dan de functie de laatste die je tegenkomt geven.

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 heb je ook nog *formele* en *actuele* parameters. Leg deze zes begrippen duidelijk uit.
b. Gegeven een C++-programma dat de volgende functies bevat:

```
int pepijn (int a, int b) {
  a = a-b+1; b *= 2; return (a+b+1);
}//pepijn
void merijn (int p, int & q) {
  int x = 8; p = p+5; x--; y = y + pepijn (p,p); x--;
  q = pepijn (q,p) + pepijn (q,y) + x;
  cout << "merijn: " << x << "," << p << "," << q << "," << y << endl;
}//merijn
```

Laten verder de globale variabelen `x`, `y` en `z` (alle van type `int`) gegeven zijn. Voor aanroep van de functie `merijn` hebben zij de waarden 5, 4 en 3 respectievelijk. Wat is dan de uitvoer van (leg je antwoord duidelijk uit):

```
merijn (x,z);
cout << "frodo: " << x << "," << y << "," << z << endl;
```

c. Dezelfde vraag als **b**, maar nu voegen we nog een `&` toe aan de parameter `p` in de heading van `merijn` en aan de parameter `b` in die van `pepijn`.
d. Dezelfde vraag als **b**, maar nu staat bij *elke* parameter een `&`. Waarom zijn er verschillende antwoorden mogelijk? Geef deze.

e. Stel dat je in `pepijn` een aanroep van `merijn` wilt doen. Mag dit en hoe krijg je dit voor elkaar in C++? Is er dan sprake van recursie?

3. We hebben twee 2-dimensionale arrays `Puz1` en `Puz2` met `m` rijen en `n` kolommen, gevuld met gehele getallen tussen 0 en 20: `int Puz1[m][n]; int Puz2[m][n];`, bijvoorbeeld:

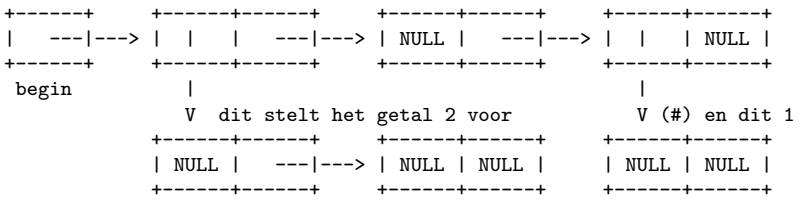
```
8  7  8  5   Puz1      7  8  5 11   Puz2     const int m = 3, n = 4;
1  2 11  5             1  2  7  5
5  8  2  7             8  8  2  5
```

a. Geef een C++-functie `int aantal (Puz1,X)` die oplevert hoe vaak de `int X` in `Puz1` voorkomt (in het voorbeeld: 5 komt 3 keer voor). Vul zelf de heading goed in!
b. Geef een C++-functie `bool perm (Puz1,Puz2)` die bepaalt of `Puz1` en `Puz2` precies dezelfde getallen bevatten — en ook nog even vaak. Voor het voorbeeld: `true`. Tip: **a**.
c. We spelen het volgende spel. Begin ergens in `Puz1`, zeg op plek (i, j); ga nu naar $(i+1, j+1)$ in `Puz2` *mits* op die plek daar hetzelfde getal staat (anders stopt het spel) en *mits* je het array niet uit loopt (anders stopt het spel); vervolgens analoog van `Puz2` naar `Puz1`, enzovoorts. Bijvoorbeeld `Puz1[0][1]` $\rightarrow$ `Puz2[1][2]` $\rightarrow$ `Puz1[2][3]`, 2 stappen. Schrijf nu een C++-functie die het grootst mogelijke aantal stappen oplevert, wanneer je alle beginplekken in `Puz1` probeert.

4. We willen bij deze opgave rijtjes getallen met behulp van pointers (op omslachtige wijze) opslaan. De pointer `begin` van type `getal*` wijst naar 't eerste getal van de lijst.

```
class getal { public: // een struct mag ook
  getal* inhoud;      // pointer naar het voorgestelde getal
  getal* volgende;    // pointer naar het volgende "getal"
};// getal
```

De getallen worden voorgesteld door het *aantal elementen* van de lijst waar de `inhoud`-pointers naar wijzen. In die lijstjes zijn steeds alle `inhoud`-pointers `NULL`. Het rijtje getallen $2, 0, 1$ gaat er uit zien als:



a. Schrijf een C++-functie `voegtoe (begin,X)` die een nieuw vakje voor in de lijst met `begin` als ingang toevoegt. De `int X` is 0 of 1; als het 1 is moet er (net als bij `(#)`) een extra vakje worden aangemaakt.
b. Schrijf een C++-functie `gooiweg (begin)` die het eerste getal uit de lijst vernietigt — mits de lijst niet leeg is. Neem aan dat het voorgestelde getal 0 of 1 is.
c. Schrijf een C++-functie `hoogop (begin)` die het eerste getal (neem aan dat de lijst niet leeg is) met 1 ophoogt.
NB De voorbeeldlijst kun je uit een oorspronkelijk lege lijst maken door middel van:

```
voegtoe (begin,1); voegtoe (begin,0); voegtoe (begin,1); hoogop (begin);
```

d. Schrijf een C++-functie `int som (begin)` die de som van de voorgestelde getallen (0 als de lijst leeg is) oplevert. In het voorbeeld: $2 + 0 + 1 = 3$.
e. In de functies bij **a**, **b**, **c** en **d** staat in de heading de parameter `begin`. Deze heb je call by value óf call by reference doorgegeven (met een `&`). Waarom, en maakt het voor de werking van deze functies verschil uit of die `&` erbij staat?