Tentamen Programmeermethoden
Donderdag 16 maart 2017, 14:00–17:00 uur
Universiteit Leiden — Informatica

Bij alle functies moeten de variabelen (constanten eventueel uitgezonderd) in de heading of locaal voorkomen; vul zelf headings goed in. De te behalen punten (totaal 100) staan tussen haakjes bij de opgaven. Succes!

1. (25) In een array `int A[n]` staan `n` (een oneven `const` $\geq$ `1`) verschillende gehele getallen.
a. (5) Schrijf een `int` C++-functie `groter (A,i,n)` die oplevert hoeveel array-elementen van `A` echt groter zijn dan `A[i]` (`A[i]` zelf dus niet meegeteld); neem aan dat $0 \leq$ `i` $<$ `n`.
b. (7) Schrijf een `int` C++-functie `mediaan (A,n)` die de mediaan van de getallen uit `A` oplevert. Gebruik de functie van **a**. De *mediaan* van een rij getallen is de waarde uit de rij waarvoor geldt dat er precies evenveel grotere als kleinere in de rij zitten.
c. (10) Schrijf een C++-functie `sorteer (A,n)` die het array `A` als volgt oplopend sorteert. Loop het array af, en wissel telkens herhaald het `i`-de element naar de juiste plek (via de functie van **a**) totdat op positie `i` het correcte element staat.
d. (3) De vergelijkingen tussen array-elementen die het algoritme van **c** doet zitten verstopt in functie-aanroepen naar de functie uit **a**. Hoeveel vergelijkingen betreft het minimaal, en wanneer gebeurt dit? Is dit minder dan / evenveel als / meer dan bij bubblesort?

2. (25) **a**. (6) 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. (6) Gegeven een C++-programma met daarin de volgende twee functies:

```
double sybrand (double & r, double & s) {
  r = s - r; s = s - r; r = r + s; return r - s; }//sybrand
double mark (double & r, double & s) {
  double temp = 7.0; int i = 4;
  for ( i = 0; i < 4; i++ ) {
    temp += sybrand (s,r); cout << i << r << s << temp << endl; }//for
  return temp; }//mark
```

Verder zijn de globale variabelen `x` en `y` van type `double` gegeven. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x = 1.0; y = 3.0; cout << mark (x,y) << endl; cout << x << y << endl;
```

c. (5) Idem, maar nu zonder de vier `&`'s in de headings van `sybrand` en `mark`.
d. (4) Wat levert op (maar nu weer met `&`-s voor de parameters in `sybrand` en `mark`):

```
x = 4.0; cout << mark (x,x) << endl; cout << x << endl;
```

e. (4) Geef twee veel eenvoudiger functies `mark2` die dezelfde return-waarde opleveren als `mark` voor alle parameters `r` en `s`, voor het geval *met* en het geval *zonder* de vier `&`'s.

3. (25) Gegeven is het twee-dimensionale array `int afstand[n][n];`, met zekere `const int n` ≥ 2. Er geldt dat `afstand`[i][j] > 0 de afstand is tussen de plaatsen `i` en `j` met `i` $\neq$ `j`, waarbij `afstand[i][i]` `0` is, en de afstand tussen `i` en `j` even groot is als die tussen `j` en `i`. Een voorbeeld met `n = 4` staat hiernaast.

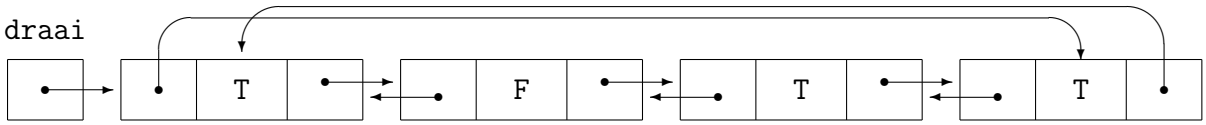
0	3	7	9
3	0	4	14
7	4	0	8
9	14	8	0

a. (7) Schrijf een C++-functie `bool reis (afstand,km)` die kijkt of er een rondreis van `i` naar `j` naar `k` naar `i` is (voor willekeurige onderling verschillende plaatsen `i`, `j` en `k`) die in totaal precies afstand `km` heeft. In het voorbeeld zou voor `26` het antwoord `true` zijn: van `0` naar `1` naar `3` naar `0` (dat is `3 + 14 + 9 = 26`).
b. (8) Schrijf een C++-functie `int verste (afstand,i)` die het nummer van de verst van `i` afgelegen plaats oplevert. In het voorbeeld, met `i = 3`, is dat `1` (wegens afstand `14`). Als er meerdere plaatsen voldoen: die met de grootste index. Neem aan dat `0` $\leq$ `i` $<$ `n`.
c. (10) Schrijf een C++-functie `int hoever (afstand,i)` die bepaalt hoeveel je in totaal reist als je begint in `i`, dan steeds naar de verst gelegen plaats gaat, en stopt zodra je ergens komt waar je al eerder geweest was. In het voorbeeld, met `i = 0`, is het antwoord `9 + 14 + 14 = 37` (van `0` naar `3` naar `1` naar `3`). Neem aan dat `0` $\leq$ `i` $<$ `n`. Hint: gebruik een Booleaans hulparray.

4. (25) Gegeven is het volgende type:

```
class paard { public: bool bezet; paard* vorige; paard* volgende; };
```

Met behulp hiervan kan een cirkelvormige dubbelverbonden lijst van paarden worden opgebouwd, bestaande uit vakjes met een Booleaanse waarde, en twee pointers naar respectievelijk vorige en volgende paard. Het eerste paard (waar de ingangspointer naar wijst) heeft als voorganger het laatste, het laatste heeft als opvolger het eerste. Als er slechts één paard is, is dit zijn eigen opvolger en voorganger. Voorbeeld (`draai`(molen) van type `paard*`):



a. (6) Schrijf een C++-functie `verwijder (draai)` die het `paard`-object uit de structuur dat door `draai` van type `paard*` wordt aangewezen, netjes verwijdert, indien het niet bezet is. Controleer of er minstens één paard is.
b. (5) Schrijf een C++-functie `voegtoe (draai,paardbez)` die een nieuw `paard`-object met `paardbez` als bezetting erin in de niet-lege structuur met ingang `draai` toevoegt. De pointer `draai` moet naar het nieuwe paard gaan wijzen.
c. (4) Schrijf een C++-functie `wissel (draai)` die de bezetting van de twee voorste paarden omwisselt. Controleer of de lijst niet leeg is.
d. (4) 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? Mag het, moet het? Leg duidelijk uit.
e. (6) Schrijf een C++-functie `draaiom (draai)` die alle pointers in de (eventueel lege) lijst omdraait: elke `vorige`-pointer moet naar het oorspronkelijk volgende paard gaan wijzen, en elke `volgende`-pointer moet naar het oorspronkelijk vorige paard gaan wijzen.