Tentamen Programmeermethoden
Maandag 6 augustus 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 een array `int A[n]` stoppen we `n` (een `const` > 0) gehele getallen.
a. Een *run* is een aaneengesloten serie array-elementen, waarbij elk element 1 verschilt met zijn voorganger. Schrijf een C++-functie `bool run (A,i,j)` die bepaalt of de array-elementen `A[i]`, `A[i+1]`,...,`A[j]` een run vormen. Neem aan dat $0 \leq$ `i` $\leq$ `j` $<$ `n`. Als `i` $=$ `j` is het antwoord altijd `true`. Voorbeeld: met array-waarden `3 7 6 7 8 4 1` (`n` $= 7$) geeft `run (A,1,4)` als resultaat `true` (wegens `7 6 7 8`, een run van lengte 4).
b. Schrijf een niet noodzakelijk efficiënte C++-functie `int longrun (A,n)` die de lengte van de (of liever: een) langste run uit `A` bepaalt. Hierbij mag `A` alleen via de functie van **a** benaderd worden.
c. Schrijf een C++-functie `int longrun2 (A,n)` die hetzelfde resultaat als **b** oplevert, maar nu efficiënt: het array moet één maal doorlopen worden, waarbij de lengte van de huidige run wordt bijgehouden evenals de lengte van de langste tot dan toe.
d. Schrijf een C++-functie `void sorteer (A,n)` die `A` *aflopend* sorteert. Gebruik een sorteermethode naar keuze.

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 gerard (bool & b, int & x) {
  b = !b; if ( b ) { x += 5; y = 6; }
  cout << b << x << y << endl; return x;
}  // gerard
int andre (bool & b, int & x) {
  b = ( x > 5 ); x += 4; y = gerard (b,x);
  cout << b << x << y << endl; return x;
}  // andre
```

Wat levert op (met globale variabelen `y` en `z` van type `int` en `waar` van type `bool`):

```
y = 2; z = 3; waar = true;
cout << andre (waar,z); cout << y << z << waar << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg.
c. Idem, maar nu zonder de vier `&`'s in de headings van `gerard` en `andre`.
d. Wat levert op (maar nu weer met `&`-s voor de parameters in `gerard` en `andre`):

```
y = 3; waar = false;
cout << andre (waar,y); cout << y << waar << endl;
```

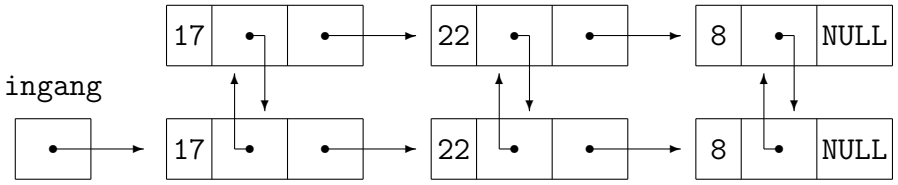
e. Mag ergens in het programma `y = gerard (waar, andre (waar,y));` staan?

3. Gegeven is een `m` bij `n` (beide `const`) array `hoogte`, gevuld met verschillende positieve gehele getallen; `hoogte[i][j]` stelt de hoogte op het punt met coördinaten (`i`, `j`) voor.
a. Schrijf een C++-functie `int hoogste (hoogte,i,j)` die in (`i`, `j`) de coördinaten van het hoogste punt oplevert, en de hoogte daarvan retourneert.
b. Stel dat je alleen omlaag kunt lopen, dat wil zeggen naar horizontaal of verticaal aangrenzende punten met kleinere hoogte. Schrijf een functie `bool kan (hoogte,i,j,r,s)` die precies `true` is als (*i*) (`i`, `j`) en (`r`, `s`) zich in dezelfde rij of kolom bevinden *en* (*ii*) je vanuit (`i`, `j`) komend en in hun gemeenschappelijke rij of kolom naar (`r`, `s`) gaand steeds lagere waardes tegenkomt.
c. Schrijf een C++-functie `int tel (hoogte)` die telt hoeveel punten je vanuit het hoogste punt kunt bereiken door herhaald van richting wijzigend omlaag te lopen. Je moet dus vanuit het hoogste punt naar een naburig punt met kleinere waarde gaan, en dan weer naar een "lager" punt, etcetera.
Hint: maak een array `bool bereikt[m][n]` waarin wordt bijgehouden welke punten direct of indirect vanuit het hoogste punt te bereiken zijn; gebruik eerst **a** en dan herhaald **b**, net zolang als nodig.

4. Gegeven is het volgende type:

```
class info { public: int getal; info* vert; info* hori; };
```

Met behulp hiervan worden rijtjes (lijstjes) met getallen opgebouwd. Het veld `hori` bevat een pointer naar het volgende `info`-object (of `NULL`), het veld `vert` een pointer naar een `info`-object erboven of eronder met dezelfde `getal`-waarde. We kunnen zo twee "dezelfde" lijsten boven elkaar maken. Een voorbeeld (`ingang` van type `info*`):



We nemen aan dat de vakjes onder en boven steeds gelijk op gaan, als in het voorbeeld.
a. Schrijf een C++-functie `voegtoe (ingang,get)` die boven elkaar twee nieuwe objecten met getal `get` erin vooraan de structuur (met `ingang` van type `info*` als ingang) toevoegt. In het voorbeeld is zojuist `voegtoe (ingang,17)` aangeroepen.
b. Schrijf een C++-functie `verwijder (ingang)` die het eerste tweetal boven elkaar gelegen objecten uit de lijst (met `ingang` van type `info*` als ingang) verwijdert, mits deze waarde even is. Denk aan de lege lijst.
c. Schrijf een C++-functie `verwissel (ingang)` die de `getal`-waardes van eerste en tweede (via de `hori`-pointer) object verwisselt indien deze bestaan, in "beide" lijsten. In het voorbeeld moeten twee maal `17` en `22` verwisseld worden.
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 C++-functie `bool check (ingang)` die controleert of de `getal`-waardes steeds boven en onder inderdaad aan elkaar gelijk zijn, zoals in het voorbeeld.