Tentamen Programmeermethoden
Maandag 1 augustus 2011, 14.00–17.00 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 `const` > 1) verschillende gehele leeftijden ≥ 0.
a. Schrijf een C++-functie `int jongst (A,n)` die de kleinste leeftijd uit `A` oplevert.
b. Schrijf een C++-functie `void twee (A,X,jonger,ouder,n)` die de hoogste leeftijd kleiner dan de integer `X` (dus $<$ `X`) in `jonger` en de laagste leeftijd groter dan `X` (dus $>$ `X`) in `ouder` deponeert. Als een van de twee niet bestaat moet daar `-1` in worden gezet. Loop hiertoe één maal door `A` heen.
c. Schijf een nieuwe versie `jongst2` van de functie `jongst` van **a**, die (elementen van) `A` alleen benadert door precies één maal de functie `twee` van **b** aan te roepen.
d. Schrijf een C++-functie `void sorteer (A,B,n)` die `A` oplopend naar het array `B` sorteert, waarbij `A` alleen benaderd mag worden door (herhaald) de functie `twee` van **b** aan te roepen.
e. Hoeveel vergelijkingen met array-elementen worden er bij **d** (ongeveer) gedaan?

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 functies:

```
int gilbert (int & a, int & b) {
  int hulp = a; a = b; b = hulp; cout << a << b << hulp << endl;
  return a + b;  }//gilbert
int george (int & a, int & b, int & x) {
  int aantal = 0;
  while ( x > 0 ) { aantal += gilbert (a,b); x--;  }//while
  cout << a << b << x << aantal << endl; return aantal;  }//george
```

De variabelen `a`, `c` en `aantal` zijn globaal en van type `int`. Wat wordt er afgedrukt door het volgende stukje programma (leg je antwoord duidelijk uit):

```
a = 5; c = 2; aantal = 5;
cout << george (a, c, aantal); cout << a << c << aantal << endl;
```

c. Als **b**, maar nu met alle vijf `&`'s weggelaten.
d. Weer terug naar de situatie van **b**, met de vijf `&`'s erbij. Wat gebeurt er bij uitvoering van het volgende stukje C++? Wat wordt er afgedrukt?

```
c = 6; cout << george (c, c, c) << endl; cout << c << endl;
```

e. En wat levert `george (a, b, x)` in het algemeen op als returnwaarde — als functie van integers `a`, `b` en `x`? Dit voor de situatie van **c**, dus met alle vijf `&`'s weggelaten.
f. Mag een statement als `a = george (gilbert (a, c), gilbert (c, a), a);` ergens in het programma staan? En zo ja, is er dan sprake van recursie?

3. Gegeven is een `m` bij `n` (beide `const` > 0) Booleaans array `huis`. In het voorbeeld rechts geldt `m` $= 3$ en `n` $= 6$. Hierbij staat `T` voor `true` en `F` voor `false`.

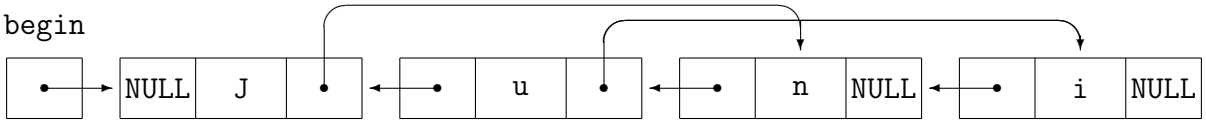
```
T F F T T T
T T F F F T
F F F T F T
```

a. Schrijf een C++-functie `telze (huis,rij)` die telt hoevaak een `F` direct tussen twee `T`'s staat in rij `rij`, met $0 \leq$ `rij` $<$ `m`. Voor `rij` $= 2$ is dit 1.
b. Schrijf een Booleaanse C++-functie `doorzichtig (huis,kol)` die bepaalt of kolom `kol`, met $0 \leq$ `kol` $<$ `n`, *doorzichtig* is, dat wil zeggen geheel uit `F`'s bestaat. In het voorbeeld: alleen `true` voor `kol` $= 2$.
c. We lopen als volgt over een *pad* door het array: bij een `T` moet je naar rechts, bij een `F` naar beneden; het pad eindigt als je het array uitloopt. Schrijf een C++-functie `int lang (huis,p,q)` die de startplek $(\texttt{p}, \texttt{q})$ bepaalt, met $0 \leq$ `p` $<$ `m` en $0 \leq$ `q` $<$ `n`, waar het (of preciezer: een) *langste* pad begint. De lengte daarvan, het aantal bezochte array-elementen, moet worden teruggegeven. In het voorbeeld: $(\texttt{p}, \texttt{q}) = (\texttt{0}, \texttt{0})$ met lengte 5. Als er meerdere langste paden zijn, mag een van de beginplekken naar keuze worden gegeven.

4. Gegeven is het volgende type:

```
class iets { public: iets* vorig; char let; iets* rechts2; };
```

Met behulp hiervan worden lijstjes met een *even* aantal objecten, en dus letters, opgebouwd. Het veld `rechts2` bevat een pointer naar het er *twee* rechts naast gelegen `iets`-object, en de `vorige`-pointer wijst naar het er direct links van gelegen `iets`-object (of `NULL`). Een voorbeeld (`begin` van type `iets*`):



We nemen bij alle onderdelen aan dat bij aanroep van de functie er reeds *minstens* 4 objecten in de lijst zitten. (Als er namelijk maar 2 in zouden zitten, kun je het tweede object niet bereiken.)
a. Schrijf een C++-functie `voegtoe (begin,let1,let2)` die *twee* nieuwe `iets`-objecten met letters `let1` en `let2` erin vooraan de lijst (met `begin` van type `iets*` als ingang) toevoegt. In het voorbeeld zijn zojuist objecten met `J` en `u` erin voorgevoegd.
b. Schrijf een C++-functie `verwissel (begin)` die de twee eerste `iets`-objecten uit de lijst `begin` verwisselt. In het voorbeeld moeten de complete objecten met `J` en `u` erin verwisseld worden.
c. Schrijf een C++-functie `verwijder (begin)` die de twee eerste `iets`-objecten uit de lijst `begin` verwijdert.
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.
En wordt het bij **b** anders als alleen de inhouden van de objecten verwisseld moeten worden?
e. Men besluit om de `rechts2`-pointers toch maar liever naar de directe rechterburen te laten wijzen. Schrijf een C++-functie `repareer (begin)` die dit tot stand brengt