Tentamen Programmeermethoden
Dinsdag 3 januari 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! Cijfers: `www.liacs.leidenuniv.nl/~kosterswa/pm/cijf/res.html`.

1. (25) In een array `int A[n]` staan `n` (een `const` ≥ 1) verschillende gehele getallen $\neq 0$.
a. (4) Schrijf een C++-functie `int nega (A,n)` die het aantal negatieve (< 0) getallen uit `A` teruggeeft.
b. (6) Schrijf een C++-functie `int eerste (A,i,n)` die de index van het eerste positieve (> 0) getal uit `A` teruggeeft met index $>$ `i`, voor een gegeven geheel getal `i`. Als zo'n getal niet bestaat moet `-1` worden geretourneerd. Denk aan de situatie `i` ≤ -1.
c. (4) Schrijf een C++-functie `busort (A,n)` die het array `A` oplopend sorteert. Geef de meest elementaire versie van *bubblesort*, waarbij in iedere ronde alle array-elementen (behalve het laatste) met hun directe buurman worden vergeleken en zo nodig verwisseld.
d. (8) Schrijf een C++-functie `raresort (A,n)` die het array `A` enigszins sorteert, en wel als volgt. Het doel is dat de positieve getallen onderling oplopend gesorteerd worden, terwijl de negatieve getallen op hun originele plaatsen blijven staan. Pas de functie van **c** zo aan dat, met behulp van de functie van **b**, in iedere ronde alleen de positieve getallen bekeken worden. Voorbeeld: `8 -1 2 3 -7 -4 4` wordt `2 -1 3 4 -7 -4 8`.
e. (3) Gegeven een rijtje met g negatieve getallen (`0` $\leq g \leq$ `n`). Hoeveel rondes in **d** zijn in het algemeen *echt* nodig, uitgedrukt in `n` en g?

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:

```
char lodewijk (char c, int m) {
  c = (char) ( ( c + m - 'a' + 'A' ) % 256 ); m--;
  if ( 'A' <= c && c <= 'Z' ) return c; else return '$'; }//lodewijk
char mark (char k, int m) {
  while ( m < 5 ) {
    cout << lodewijk (k,m) << endl; m += 2; }//while
  cout << k << "," << m << endl; return lodewijk (k,m); }//mark
```

Verder zijn de globale variabelen `a` van type `char` en `m` van type `int` gegeven. Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
a = 'd'; m = 2; cout << mark (a,m) << endl;
cout << a << "," << m << endl;
```

c. (5) We voegen nu een `&` toe bij beide parameters `m` (dus niet bij `c` en `k`) in de heading van `lodewijk` en `mark`. Beantwoord opnieuw vraag **b**.
d. (4) Als in de functie `lodewijk` ergens `m = (int) mark ('&',m-42);` staat, compileert het programma dan nog? Onderscheid gevallen met en zonder `&`.
e. (4) Geef een for-loop waarin 26 opeenvolgende aanroepen van `lodewijk` als resultaat `ABC...Z` afdrukken. Het moet werken zowel met als zonder `&` bij de parameter `m`.

3. (25) Gegeven is een `m` bij `n` (beide `const` $>$ `1`) array `M` met gehele getallen ≥ 0. Zie hiernaast voor een voorbeeld met `m` $= 4$ en `n` $= 5$. De constantes `m` en `n` hoeven bij deze opgave niet doorgegeven te worden als parameter. Een rij/kolom heeft de *Sudoku-eigenschap* als deze (afgezien van eventuele nullen) *verschillende* getallen bevat. Een *Sudoku* is een array waarvan alle rijen en kolommen de Sudoku-eigenschap hebben.

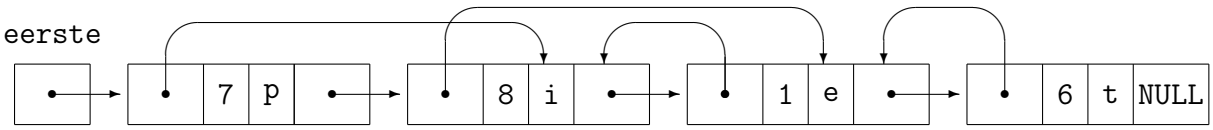
3	0	4	20	0
7	12	4	4	2
1	2	0	4	1
6	9	3	0	0

a. (7) Schrijf een Booleaanse C++-functie `checkrij (M,i)` die controleert of de `i`-de rij van `M` de Sudoku-eigenschap heeft. Voorbeeld: `true` voor rij `0` en `3`, `false` voor `1` en `2`.
b. (8) Neem aan dat `M` een Sudoku is. Schrijf een C++-functie `losop (M)` die alle nullen door getallen > 0 vervangt, waarbij steeds het kleinst mogelijke getal moet worden ingevuld zodat `M` een Sudoku blijft. Loop hierbij rij voor rij, en per rij van links naar rechts. Het hoeft niet efficiënt. Een functie `checkkolom`, analoog aan `checkrij`, mag gebruikt worden.
c. (10) Neem aan dat `M` een Sudoku is zonder nullen. We lopen als volgt door het array `M`. Start bij $(\texttt{i}, \texttt{j})$. Ga naar de grootste van de één of twee horizontaal aangrenzende directe buren, daarna naar de grootste van de één of twee verticaal aangrenzende directe buren, dan weer horizontaal, etcetera. De wandeling stopt als je ergens komt waar je al eerder bent geweest. Schrijf een C++-functie `int aantal (M,i,j)` die het aantal stappen uitrekent; in `i` en `j` moeten de coördinaten van het laatst bezochte punt komen. In het voorbeeld (waarin overigens nullen staan): beginnend bij $(\texttt{1}, \texttt{2})$ eerst links, dan omlaag, links, omhoog, rechts, en klaar; antwoord `5`, en $(\texttt{i}, \texttt{j}) = (1, 1)$. Hint: gebruik een Booleaans hulparray.

4. (25) Gegeven is het volgende type:

```
class vak { public: vak* ander; int cijfer; char naam; vak* volg; };
```

Hiermee wordt een lijst van vakken gemaakt; `naam` is een kleine letter. Het veld `volg` bevat een pointer naar het volgende `vak`-object, en `ander` naar het vorige *of* volgende (zulks naar keuze van de gebruiker), *of* `NULL` als die er beide niet zijn (en alleen in dat geval). Voorbeeld, met `eerste` van type `vak*` en `volg`-pointers horizontaal getekend:



a. (5) Schrijf een C++-functie `voegtoe (eerste,vakcijfer,vaknaam)` die een nieuw `vak`-object met `vakcijfer` en `vaknaam` erin vooraan in de lijst met ingang `eerste` toevoegt. Zet de `ander`-pointer van het oude voorste object (als dat bestaat, en die pointer `NULL` is) ook goed.
b. (5) Schrijf een C++-functie `verwijder (eerste)` die het voorste `vak`-object uit de structuur die door `eerste` wordt aangewezen, netjes verwijdert — mits het bestaat. Zet een eventuele `ander`-pointer die er naar wijst ook goed.
c. (5) Schrijf een C++-functie `wissel (eerste)` die de vak-cijfers van de twee voorste vakken omwisselt, mits deze bestaan en minstens één van deze `'p'` als naam heeft (in het voorbeeld: 7 en 8 worden verwisseld). Controleer of de lijst minstens twee objecten heeft.
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/zonder 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 `keerom (eerste)` die alle `ander`-pointers die naar het vorige object wijzen naar het volgende laat wijzen, en omgekeerd. De `ander`-pointers van voorste en laatste object moeten niet veranderen.