Tentamen Programmeermethoden
Maandag 6 januari 2003, 14.00–17.00 uur
Universiteit Leiden — Informatica

Bij alle te schrijven functies moeten de variabelen in de heading voorkomen (niet stiekem globale variabelen gebruiken). De opgaven tellen alle vier even zwaar mee. Veel succes!

1. Deze opgave gaat over array's met `n` integers: `int A[n];`, met `const int n = 10000;`. De arrays bevatten verschillende positieve getallen.
a. Geef een C++-functie `int zoek (A,n,X)` die met *lineair zoeken* de integer `X` in `A` opzoekt en de *array-index* `i` met `X` gelijk aan `A[i]` teruggeeft (`-1` als `X` niet voorkomt).
b. Geef een C++-functie `void draaien (A,n)` die de elementen in `A` als volgt "draait": alle array-elementen schuiven één stapje naar links, en het eerste array-element wordt het laatste. Zo wordt `31 41 59 27` nu: `41 59 27 31`.
c. Geef een C++-functie `int eennakleinste (A,n)` die het op-een-na-kleinste array-element van het array `A` oplevert. Het array mag alleen maar met behulp van de functie `zoek` van onderdeel **a** benaderd worden.
Hint: zit `0` in `A`? Zit `1` in `A`?

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. Een zeker C++-programma bevat de volgende programmaregels:

```
int plus (int x, int y) {
  x++; y--; z++; cout << x << y << z << endl;
  return x+y+z;
}  // plus
int onzin (int x, int y) {
  int z = plus (plus (x,y),y); cout << x << y << z << endl;
  return z;
}  // onzin
```

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

```
x = 1; y = 5; z = 8;
cout << onzin (x,y) << endl;  // (*)
cout << x << y << z << endl;
```

Wat wordt er afgedrukt? Geef hierbij uiteraard uitleg.
c. Als **b**, maar nu met `&`'s voor beide `y`'s in de headings van `plus` en `onzin`. Geef aan wat het verschil is tussen de situaties waarin bij `plus (plus (x,y),y)` de door de aanroep `plus (x,y)` gewijzigde `y`, respectievelijk de oorspronkelijke `y` voor de tweede parameter gebruikt wordt.
d. Stel dat voor alle vier voorkomende parameters in de functie-headings een `&` wordt gezet. Wat gebeurt er nu?
e. Als **b**, dus zonder de `&`'s, maar met `cout << onzin (z,z) << endl;` in plaats van de regel met `(*)`.

3. Een zoekpuzzel `char puzzel[m][n]` bestaat uit `m` rijen en `n` kolommen waarin woorden verborgen zitten. In deze opgave geldt de regel dat woorden horizontaal en verticaal staan en alleen van links naar rechts en van boven naar beneden. Een voorbeeld:

```
r v z . .
o q a a p
o p s l e
s b a n w
```

Hierin zitten bijvoorbeeld de woorden `op`, `roos`, `aap`, en `ban`. Nog niet bezette plaatsen in de puzzel worden met een `.` (punt) aangeduid. In het vervolg van deze opgave zijn we geinteresseerd in het *maken* van dit soort puzzels. Ga er vanuit dat je gebruik kunt maken van een functie `int random ( )` die een willekeurige positieve integer teruggeeft.
a. Schrijf een functie `bool randomplaats (puzzel,rij,kol)` die de random-functie gebruikt om een willekeurige plaats `(rij,kol)` in de puzzel te vinden die een `.` (punt) bevat. Als dit na 10000 pogingen nog niet gelukt is moet de functie stoppen en `false` teruggeven — en anders `true`; in dat laatste geval bevatten `rij` en `kol` de gevonden plaats.
b. Schrijf een functie `bool plaatswoord (puzzel,rij,kol,woord,wlen)` die een gegeven woord (`char woord[ ]` met lengte `wlen`) *horizontaal* neerzet in de puzzelmatrix op beginpositie `(rij,kol)` — mits dit geen conflicten met reeds geplaatste woorden geeft. Als die er wel zijn, mag de puzzel niet gewijzigd worden. De functie levert als resultaat `true` als het woord geplaatst is, en anders `false`. Voorbeeld: stel dat we `plaatswoord` aanroepen met `rij=2`, `kol=1` en `woord=roos` voor de onderstaande deels ingevulde puzzel:

```
  0 1 2 3 4
0 . . . . .
1 . . s a p
2 . . o . .
3 . . p . .
4 . . . . .
```

In dit geval kan het woord `roos` geplaatst worden en dient `plaatswoord` als resultaat `true` op te leveren. Voor `rij=3` en `kol=1` zou het resultaat `false` zijn.
c. Schrijf een functie `bool vindwoord (puzzel,woord,wlen)` die karakter voor karakter controleert of een gegeven woord *ergens* in de matrix `puzzel` *horizontaal en/of verticaal* is terug te vinden, en `true` oplevert als dat zo is — en anders `false`.

4. Gollem staat aan de rand van een moerassig gebied, Aan de andere kant van het moeras bevindt zich zijn geliefde ring. Hij moet door het moeras om de ring te halen. Onderweg houdt Gollem bij wat voor terrein hij oversteekt (zodat hij later gemakkelijk de weg terug kan vinden). Het gebied bestaat uit de volgende typen terrein: drijfzand (type 1), giftig dampend water (type 2) en veilige rotsen (type 3). Voor de opgave kun je gebruik maken van de volgende klasse-definities:

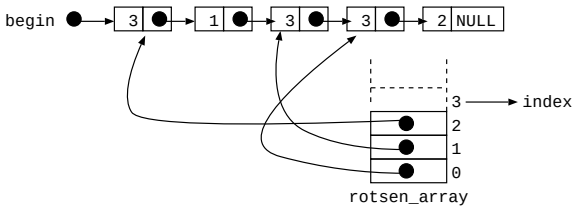
```
class terrein_element {
  public:
    int type; // terrein type
    terein_element *volgende;
};  // terrein_element
```

```
class route {
  public:
    terrein_element *begin;
    void voeg_terrein_toe (int terrein);
};  // route
```

a. Gollem roept `voeg_terrein_toe (...)` aan, iedere keer als hij een nieuw terrein betreedt (zie ook de bovenste lijst in Figuur 1). Maak voor Gollem de klasse `route` af door de methode `void voeg_terrein_toe (int terrein)` te implementeren. Het nieuwe vakje moet *vooraan* de "`route`-lijst" worden toegevoegd.

Wanneer Gollem de ring heeft bereikt heeft hij een route-beschrijving die hem in staat stelt om dezelfde tocht terug te maken. Maar Gollem is niet gek: hij wil liever van rots tot rots springen en niet de stukken drijfzand en water doorwaden (daar wordt hij ziek van).

Daarom houdt hij een *array van pointers* bij waarbij de pointers wijzen naar de veilige rotsen (zie ook Figuur 1). Concreet: iedere keer als hij een element toevoegt aan zijn "`route`-lijst" en het is type 3 (een rots), dan bewaart hij het adres van dit element in het array.



Figuur 1: Gollem's rotsen array

b. We noemen het array `rotsen_array` en gaan ervoor zorgen dat dit gevuld wordt zoals aangegeven in de figuur. We gebruiken een index, die aangeeft waar de nieuwe pointer moet komen. Voor het vullen van het array maakt Gollem gebruik van een functie `void push (...)`. Schrijf deze functie `push` met de juiste parameters. Ga er vanuit dat er altijd genoeg ruimte in het array is.

c. De functie `terrein_element* pop (...)` levert het adres op van de "bovenste" rots uit het array en verlaagt de index. Als er geen rotsen meer zijn moet de functie `NULL` retourneren. Schrijf deze functie `pop`.

Eenmaal bij de ring aangekomen kan hij een `pop` doen om de eerste rots op de terugweg te vinden, en weer een `pop` voor de volgende, enzovoorts, net zo lang tot hij weer veilig terug is.

d. Stel dat Gollem bij de ring is aangekomen en zowel de route uit **a** als het array van pointers uit **b** en **c** gevuld zijn. Schrijf een functie `sprongen` die gebruik maakt van de `pop` functie om erachter te komen hoeveel van de rotsen direct vooraf worden gegaan door andere rotsen (in Figuur 1 is dat bijvoorbeeld één keer). Let op: je mag voor deze opgave niet vanaf `begin` naar het eind van de lijst lopen. Je *moet* "terugspringen" met behulp van het array van pointers.