



# Enkele aanwijzingen

## Stap 1.

Schrijf eerst een programma dat de gebruiker en de computer om de beurt een zet laat doen, waarbij de computer random zet. Een zet van de gebruiker wordt gegeven in de vorm  $(i,j)$ . Er moet altijd gecontroleerd worden of een ingevoerde zet een geldige zet is. Zo niet dan moet de speler een nieuwe zet invoeren. Het programma moet na elke zet controleren of een van beide spelers al heeft gewonnen en zo ja wie (de winnaar wordt afgedrukt en het spel is afgelopen). Ook als een remisestand bereikt is moet dit gemeld worden. Schrijf hiervoor eerst een functie die het aantal witte stenen – het aantal zwarte stenen berekent. Deze functie kan dan later (Stap 4) nogmaals gebruikt worden. Voor het bepalen van de random zet wordt eerst berekend hoeveel toegestane zetten er mogelijk zijn, zeg `aantal`, en vervolgens wordt random bepaald welke van de `aantal` vervolgzetten de computer doet.

Schrijf een klasse `othello` die in ieder geval een 2-dimensionaal array bevat waarin het bord is opgeslagen. Dit array mag hooguit `max` bij `max` groot zijn. Een element van het array geeft een veld van het bord weer. Dit element heeft de waarde 1 als zwart daar een schijf heeft staan, 2 als wit daar een schijf heeft staan, en 0 als het veld leeg is. Bovendien moet de speler die aan de beurt is in objecten van de `othello`-klasse staan, evenals de grootte van het bord. De klasse `othello` bevat verder methoden/member-functies (eventuele parameters en return waarden moet je zelf invullen) zoals:

- `othello(...)`: de constructor (zet de beginconfiguratie op).
- `toegestanezet(...)`: controleert of een gegeven zet op plek  $(i,j)$  toegestaan is voor de speler die aan de beurt is.
- `drukaf(...)`: drukt het Othello-bord af op het scherm.
- `verschil(...)`: bepaalt het aantal witte stenen – het aantal zwarte stenen.
- `winnaar(...)`: geeft aan of het spel is afgelopen en of er een winnaar is (en zo ja wie).
- `doezet(...)`: voert een gegeven geldige zet uit.

**Stap 2.** Breid het door jou geschreven programma zodanig uit dat de speler, wanneer hij aan de beurt is, ook ervoor kan kiezen zijn laatst gedane zet (en meteen de tussenliggende computerzet) terug te nemen. Hiertoe moeten alle tussenstanden op een stapel worden bijgehouden. Dat betekent dat alle *standen*, en niet de zetten, worden onthouden. Hiermee kunnen dus ook meerdere zetten van de speler achter elkaar teruggenomen worden. Zodra de speler zet, wordt de oude stand (met het complete bord) opgeslagen.

Schrijf hiervoor een geschikte klasse `stapel`, met in elk geval memberfuncties `zetopstapel(...)` en `haalvanstapel(...)`.

### Stap 3.

Slits het programma op in een bestand waarin `main()` staat, een bestand waarin de implementatie van klasse `othello` staat, en een header file met daarin de definitie van deze klasse (zonder implementatie). Hetzelfde voor de klasse `stapel`.

Schrijf een `Makefile` voor deze files zodanig dat de `.cc` bestanden apart worden gecompileerd en uiteindelijk worden samengevoegd (“gelinkt”) tot een executeerbaar programma.

### Stap 4.

Breid het programma nu zodanig uit dat de computer behalve random zetten ook “verstandige” zetten kan doen. Een verstandige zet wordt als volgt gevonden. Vanuit een gegeven stand kunnen we alle mogelijke borden bekijken die na het doen van  $d$  zetten voorkomen: we kijken dus  $d$  zetten vooruit.

Er moet een *recursive* member-functie `eval(...)` geschreven worden die aan elke stand een zekere waarde toekent. Voor de standen die na  $d$  zetten bereikt kunnen worden (een soort “eindstanden”) wordt het aantal witte stenen – het aantal zwarte stenen bepaald. Is die waarde groot positief dan is de stand gunstig voor wit, is de waarde groot negatief, dan gunstig voor zwart. Voor de andere standen wordt de waarde bepaald uit de waarden van de directe vervolgstanden (recursie!). We doen dit als volgt. Als in een gegeven situatie wit aan de beurt is, wil die een groot positieve waarde van `eval(...)` bereiken. Hij kiest dus die zet die hem de vervolgstand met de grootste `eval(...)`-waarde zal opleveren. Die betreffende waarde wordt dan de `eval(...)`-waarde voor de gegeven stand. Voor speler zwart geldt dat hij juist de kleinste waarde kiest. Als de computer aan de beurt is wordt zijn beste zet bepaald op grond van de `eval(...)`-waarden van alle mogelijke standen na één zet. Zijn er meerdere zetten die dezelfde waarde opleveren, dan mag de computer er willekeurig een kiezen (bijvoorbeeld de eerste de beste).

Het aantal vooruit te kijken zetten  $d$  wordt door de speler ingevoerd. Let er op dat een spel soms afgelopen is voordat de diepte  $d$  bereikt wordt. Merk op dat  $d$  een parameter moet worden van de te schrijven recursive functie.

Zie voor meer uitleg ook het college van 18 november a.s.

**Stap 5.** Schrijf een grafische interface voor het spel Othello met behulp van Qt designer. Een speler moet het spel tegen de computer kunnen spelen. De computer controleert wanneer gewonnen is en door wie, etc. Geef de schijven van de ene speler een andere kleur dan die van de andere speler. Zorg voor een knop waarmee je de computer random of met verstandige zetten kan laten spelen, en voor een knop waarmee de gebruiker kan kiezen of hij een zet doet of een zet terugneemt. In het grafische programma mag je  $d$  en de grootte van het speelbord als constantes nemen. Kies een waarde voor  $d$  die voor jouw programma goed werkt. Leg verder ook vast welke speler met welke kleur speelt.

Houd je functies compact. Het liefst geen functies die langer zijn dan een scherm aan programma-code. Voorzie elke functie van commentaar. Let op goed parametergebruik: alle parameters in de heading doorgeven, en de variabele-declaraties bij het begin van mainen andere functies. De enige te gebruiken (niet zelfgemaakte) headerfile in Stap 1 t/m 4 is in principe `iostream`. Denk aan het infoblokje. Let er op dat er twee versies ingeleverd dienen te worden, een zonder het grafische interface (Stap 1 t/m 4) en een met (Stap 5).

Uiterste inleverdatum: **vrijdag 5 december 2003, 17.00 uur.**

Manier van inleveren:

1. Email aan de hoofdnakijker `pm@liacs.nl` sturen. Zorg er voor dat je *alleen* de programma-code inlevert en niet de executables. Je kunt ofwel alle bestanden apart ‘attachen’, of er (op UNIX) een enkel `tar` bestand van maken (waarschijnlijk makkelijker). Met `tar` kun je vanaf een UNIX systeem meerdere bestanden in één file stoppen. Als bijvoorbeeld je bestanden staan in directory `./opdracht4/`, dan kun je deze directory, alle bestanden in deze directory, en alle subdirectories en hun bestanden in deze directory in één `tar` bestand (bijv. genaamd `opdracht4.tar`) stoppen door middel van het volgende commando: `tar cvf opdracht4.tar opdracht4/`. Nogmaals: zorg ervoor dat deze directories *alleen* programma-code bestanden bevatten. Stuur dit `tar` bestand als attachment naar `pm@liacs.nl`.
2. En *ook* een listing (twee versies!) op papier deponeren in de daarvoor bestemde oranje doos “Programmeermethoden” in computerzaal 302/304. Overall duidelijk datum en namen van de makers vermelden, in het bijzonder in de eerste regels van de C++-code.

Te gebruiken compiler: C++: het programma moet op een Linux-machine (met `g++`) draaien.

Normering: layout 1; commentaar 2; modulariteit 2; werking zonder Qt 3; werking Qt 2.  
Eventuele aanvullingen en verbeteringen: lees de WWW-bladzijde.