Vierde programmeeropdracht: Nim

In deze opdracht leer je werken met recursie, stapels, Makefiles en Qt designer. Lees voor je aan de opdracht begint eerst de corresponderende hoofdstukken van het dictaat.
In deze opdracht moet je (1) een variant van het spel Nim programmeren (n groepjes met elk hooguit *maxLucifer* lucifers), (2) het spel in eerste instantie op een eenvoudige manier afbeelden, en (3) uiteindelijk het spelen van het spel grafisch weergeven met behulp van Qt.

In deze opdracht bestuderen we de volgende variant op het Nim-spel. De spelregels zijn eenvoudig. Op tafel liggen n (≥ 1) groepjes lucifers (hooguit $maxGroep = 20$ (bijvoorbeeld)), waarbij elk groepje maximaal *maxLucifer* lucifers bevat. Hierin zijn *maxGroep* en *maxLucifer* constanten. Er zijn twee spelers (in dit de geval de gebruiker en de computer). De spelers mogen om de beurt uit één van de groepjes 1 of 2 lucifers pakken. De speler die de allerlaatste lucifer(s) wegneemt heeft gewonnen. We spreken af dat de gebruiker altijd mag beginnen. Het spel moet een aantal keren achter elkaar gespeeld kunnen worden. Bij het begin van elk spel geeft de gebruiker het aantal groepjes n op en het aantal lucifers (ten minste 1) per groep. Tijdens het spelen kan het aantal lucifers in een groepje nul zijn.
Als de gebruiker aan zet is, kan deze speler een toegestane (=geldige) zet doen, of juist de laatste eigen zet (en meteen de tussenliggende computerzet) terugnemen. Voor dit laatste gebruiken we een stapel (stack). Een geldige zet betekent uiteraard dat één of twee lucifers worden weggenomen uit een bestaand groepje, en dat die betreffende groep ook inderdaad nog "genoeg" lucifers bevat.
De computer doet gedurende een spel ofwel steeds random zetten (optie 1), ofwel vanaf een zekere grootte (het totaal aantal lucifers) steeds winnende zetten (indien die bestaan) (optie 2). De gebruiker moet voor het spel begint opgeven op welk van de twee manieren de computer speelt. Voor het bepalen van een winnende zet gebruiken we recursie.
In eerste instantie (Stap 1 t/m 4) wordt de stand steeds in een eenvoudig formaat op het scherm getoond, later (Stap 5) wordt Qt gebruikt om een en ander te verfraaien.

Enkele aanwijzingen

Stap 1.
Schrijf eerst een programma dat de gebruiker en de computer om de beurt een zet laat doen, waarbij de computer telkens random zet. Een zet wordt gegeven in de vorm (`nummer`, `aantal`): er moeten `aantal` lucifers uit groep `nummer` worden gehaald. 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).
De random zet van de computer wordt als volgt bepaald. Eerst wordt berekend hoeveel toegestane (=geldige) zetten er mogelijk zijn, zeg `toegestaan`, en vervolgens wordt random bepaald welke van de `toegestaan` vervolgzetten de computer doet. Hier kun je bijvoorbeeld de randomgenerator uit het dictaat voor gebruiken.
Schrijf een klasse `nim` die in ieder geval een 1-dimensionaal array bevat waarin een spelsituatie als volgt is opgeslagen: de index van een arrayelement geeft het nummer van

het groepje aan en de inhoud het aantal lucifers in dat groepje. Het array is dus hooguit *maxGroep* groot. Bovendien moeten objecten van de klasse `nim` de speler die aan de beurt is bevatten, alsmede het aantal groepjes n. De klasse `nim` bevat verder methoden/memberfuncties (eventuele parameters en returnwaarden moet je zelf invullen) zoals:

- `nim(...)`: de constructor (initialiseert spelarray en overige membervariabelen).
- `beginstand(...)`: zet de door de gebruiker in te voeren beginsituatie op.
- `geldigezet(...)`: controleert of een gegeven zet (`nummer`, `aantal`) is toegestaan
- `drukaf(...)`: drukt de huidige stand af op het scherm.
- `winnaar(...)`: geeft aan of het spel is afgelopen en zo ja wie de winnaar is.
- `doezet(...)`: voert een gegeven geldige zet uit.
- `randomzet(...)`: bepaalt een random zet voor de computer.
- `winnend(...)`: geeft een winnende zet (zie Stap 4).

Stap 2. Breid het door jou geschreven programma zodanig uit dat de gebruiker, wanneer hij aan de beurt is, ook ervoor kan kiezen zijn laatst gedane zet (en meteen de tussenliggende computerzet) terug te nemen. Na het terugzetten is de speler opnieuw aan de beurt. Er kunnen zo dus ook meerdere zetten van de speler achter elkaar teruggenomen worden. Om zetten terug te kunnen nemen moeten tussenstanden op een stapel (stack) worden bijgehouden. Dat betekent dat alle voorgaande standen waarin de gebruiker aan de beurt is moeten worden onthouden. Zodra de speler zet, wordt steeds de oude stand boven op de stapel gezet. Schrijf hiervoor een geschikte klasse `stapel`, met in elk geval memberfuncties `zetopstapel(...)` en `haalvanstapel(...)`.

Stap 3.
Splits het programma op in een bestand waarin `main()` staat, een bestand waarin de implementatie van klasse `nim` 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 winnende zetten kan doen. Bovendien moet het programma wanneer de gebruiker aan de beurt is en het spel niet te groot (zie hieronder) is als hint een winnende zet geven (indien die bestaat).
Schrijf hiertoe een *recursieve* memberfunctie `winnend (...)` die voor de huidige stand een winnende zet oplevert voor de aan de beurt zijnde speler. Een zet bestaat uit het aantal lucifers `aantal` en het nummer `nummer` van de groep lucifers waaruit dat aantal moet worden weggenomen. Een winnende zet is dan uiteraard een zet die een situatie oplevert die niet winnend is voor de tegenstander. Als de gegeven toestand niet winnend is geeft de functie bijvoorbeeld als waarden $(-1, -1)$ terug.

De recursieve functie `winnend (...)` is niet erg efficiënt en een aanroep duurt voor bijvoorbeeld 10 groepjes met elk 10 lucifers al veel te lang. Daarom mag de functie *alleen* aangeroepen worden als het totaal aantal lucifers dat nog op tafel ligt minder is dan een als constante `grootte` aan het begin van je programma op te geven waarde. Ga zelf na wat een acceptabele waarde is voor de door jou geschreven functie `winnend (...)`.
Als de gebruiker bij aanvang van het spel voor optie 2 gekozen heeft, dan moet zodra het totaal aantal lucifers minder is dan `grootte`, de functie `winnend (...)` worden aangeroepen als de computer aan zet is. De computer voert dan de winnende zet uit als die bestaat, en doet anders een willekeurige zet (bijvoorbeeld de eerste de beste, of random). Zowel bij optie 1 als bij optie 2 moet vanaf het moment dat het totaal aantal lucifers minder is dan `grootte`, aan de gebruiker een voor hem winnende zet getoond worden.

Stap 5. Schrijf een grafische interface voor het spel Nim 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. We spelen het spel standaard met vier groepjes lucifers, die in de beginsituatie elk vijf lucifers bevatten. Om het spel weer te geven gebruiken we een 5 bij 4 rooster met 20 vakjes: de verschillende kolommen stellen de groepjes lucifers voor Elke kolom correspondeert zo met een groepje lucifers, en elk vakje met één enkele lucifer: een vakje is zwart als er een lucifer ligt, wit als dat niet het geval is. De aanwezige lucifers in een groep liggen altijd aaneengesloten, beginnend onderin de betreffende kolom.
Als de gebruiker op het bovenste zwarte vakje uit een kolom klikt, dan wordt één lucifer uit de betreffende kolom gehaald (de bovenste!). Drukt hij op het op een na bovenste zwarte vakje, dan worden de bovenste twee weggehaald. De andere vakjes worden niet als zet geaccepteerd. Na de zet van de gebruiker doet meteen de computer een zet.
Zorg ook voor een knop waarmee je aangeeft of de computer random of winnend speelt. Tijdens het spel kan de gebruiker door op de knop te klikken de speelwijze veranderen. Geef met een tekst in de knop aan wat de huidige speelwijze is. Deze knop kan dus op en neer toggelen tussen de twee verschillende speelwijzen. Verder moet er een knop zijn waarmee de gebruiker een zet terugneemt. Het spel hoeft niet herhaald gespeeld te kunnen worden. Er hoeft ook geen winnende zet aan de gebruiker getoond te worden.

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 `main`en 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 3 december 2004, 17.00 uur** (Stap 1 t/m 4) en **vrijdag 10 december 2004, 17.00 uur**: het geheel.

Manier van inleveren:

1. Op vrijdag 3 december moeten Stap 1 t/m 4 klaar zijn en dient het bijbehorende werkende programma te worden ingeleverd. De rest volgt een week later.

2. 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`.

3. En *ook* een listing (in totaal twee versies!) op papier deponeren in de daarvoor bestemde doos "Programmeermethoden" in de postkamer, kmr 156. Overal 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.