1 En C++-lab: quaternioner

Quaternioner introduceras, 1843, av den irländske matematikern Sir William Rowan Hamilton (1805-1865), och de är en slags generalisering av komplexa tal. För mer detaljer, se:
`http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Hamilton.html` och
`http://turnbull.mcs.st-and.ac.uk/history/Biographies/Hamilton.html`

Istället för att ha *en* imaginär enhet har vi tre, i, j, k. Dessa uppfyller sambanden:

$$i^2 = j^2 = k^2 = ijk = -1$$

En quaternion kan skrivas: $a = a_1 + a_2 i + a_3 j + a_4 k$ där a_1, a_2, a_3, a_4 är reella tal (jämför komplexa tal som kan skrivas $a_1 + a_2 i$). Vi skriver även $a = (a_1, a_2, a_3, a_4)$ (dvs som en vektor). När Hamilton kom på sambanden så:

> And here there dawned on me the notion that we must admit, in some sense, a fourth dimension of space for the purpose of calculating with triples ... An electric circuit seemed to close, and a spark flashed forth.

Han ristande in de grundläggande likheterna i en stenbro, Broome Bridge när han och hans fru passerade. Inga lämningar av ristandet finns kvar, men "Royal Irish Academy" har satt upp en minnestavla.

> Here as he walked by on the 16th of October 1843 Sir William Rowan Hamilton in a flash of genius discovered the fundamental formula for quaternion multiplication
>
> $$i^2 = j^2 = k^2 = ijk = -1.$$
>
> & cut it on a stone of this bridge.

Den som går på matematikprogrammet kommer nog att stöta på quaternioner i kursen om algebraiska strukturer.

Nu till lite matematiska detaljer.
Addition definieras som för vektorer och den är med andra ord kommutativ. Subtraktion definieras analogt. Multiplikation är inte kommutativ, det gäller t.ex. att $ij = k$ men $ji = -k$. Övning: visa detta. Det gäller även att $jk = i, kj = -i$ och $ki = j, ik = -j$.
Multiplikation av quaternioner, vilken är distributiv, kan nu skrivas, med $a = (a_1, a_2, a_3, a_4)$, $b = (b_1, b_2, b_3, b_4)$:

$$ab = a_1 b + a_2 ib + a_3 jb + a_4 kb = (a_1 b_1 + a_1 b_2 i + a_1 b_3 j + a_1 b_4 k) +$$

$$(a_2 b_1 i + a_2 b_2 i^2 + a_2 b_3 ij + a_2 b_4 ik) + (a_3 b_1 j + a_3 b_2 ji + a_3 b_3 j^2 + a_3 b_4 jk) +$$

$$(a_4 b_1 k + a_4 b_2 ki + a_4 b_3 kj + a_4 b_4 k^2) =$$

$$(a_1 b_1 - a_2 b_2 - a_3 b_3 - a_4 b_4) + (a_1 b_2 + a_2 b_1 + a_3 b_4 - a_4 b_3) i +$$

$$(a_1 b_3 - a_2 b_4 + a_3 b_1 + a_4 b_2) j + (a_1 b_4 + a_2 b_3 - a_3 b_2 + a_4 b_1) k$$

Man kan också definiera inversen, a^{-1}, till en quaternion genom att kräva att $a^{-1} a = 1$ och man ser (efter lite räknande att)

$$a^{-1} = \frac{a_1 - a_2 i - a_3 j - a_4 k}{a_1^2 + a_2^2 + a_3^2 + a_4^2}$$

Detta är även högerinversen, dvs. $a a^{-1} = 1$. Om vi inför en naturlig konjugatoperation så ser vi att

$$a^{-1} = \frac{\bar{a}}{a_1^2 + a_2^2 + a_3^2 + a_4^2}$$

och med en rimlig definition av absolutbelopp så får vi

$$a^{-1} = \frac{\bar{a}}{|a|^2}$$

eller $|a| = \sqrt{a\bar{a}}$ dvs. analogt med komplexa tal.

Quaternioner kan användas i datorgrafik för att hantera rotationer (samt t.ex. i formuleringen av Einsteins allmänna relativitetsteori). Låt q vara en quaternion av längd ett (dvs. $|q| = 1$). Man kan visa att en sådan q kan skrivas på formen $\cos(\theta/2) + \sin(\theta/2)(q_2 i + q_3 j + q_4 k)$ där den vanliga reella vektorn (q_2, q_3, q_4) har längden ett. Låt nu $r = r_2 i + r_3 j + r_3 k$, så att $r_1 = 0$ och beräkna:

$$s = qrq^{-1} = qr/q$$

med en naturlig definition av *höger*-division. (Eftersom multiplikationen inte är kommutativ så är *inte* $s = r$.) Med rätt mycket räknande kan man visa att (s_2, s_3, s_4), tolkat som en vanlig reell vektor, svarar mot en rotation av vektorn (r_2, r_3, r_4) vinkeln θ kring vektorn (q_2, q_3, q_4) (när vi vrider ser vi i q-vektorns riktning och vrider medurs).

Låt oss se på ett enkelt exempel: tag $\theta = \pi/2$, $(q_2, q_3, q_4) = (1, 0, 0)$ samt $(r_2, r_3, r_4) = (0, 1, 0)$. Resultatet blir då (kontrollera!) vektorn $(0, 0, 1)$, svarande mot quaternionen $(0, 0, 0, 1)$ (eftersom det svarar mot att rotera y-axeln $\pi/2$ runt x-axeln, vilket ger z-axeln).

Uppgiften i laborationen är att skriva ett C++-program som stödjer räkning med quaternioner. *Om* man använder klasser och överlagring av operatorer kan man få lättläst kod. Exemplet ovan kan då se ut som:

```
// Del av koden i C++
  double theta2 = M_PI / 4.0;
  Quaternion q(cos(theta2), sin(theta2), 0, 0);  // definiera q
  Quaternion r(0, 0, 1, 0);                      // definiera r

  cout << "q * r / q = " << q * r / q << endl;   // beräkna och skriv ut qr/q
...

% a.out
q * r / q = (0, 0, 1.77636e-15, 1)
```

där `1.77636e-15` är avrundningsfel. Observera att jag har definierat en speciell funktion som svarar mot `*`, quaternionmultiplikation, analogt för övriga räkneoperationer. Dessutom är `<<` överlagrad så att `q * r / q` skrivs ut på ett lämpligt sätt. Nu skulle kursen behöva vara några veckor längre för att vi skulle hinna ta upp klasser, så du för göra en enklare lösning. Gör följande:

- Det är lämpligt att använda en post för att lagra en quaternion. Du får själv fylla i detaljerna.
- Implementera följande beräkningsrutiner i C++, (`q1`, `q2` och `q` är tre quaternionvariabler, poster):
  1. `q = qadd(q1, q2)` som beräknar `q = q1 + q2`
  2. `q = qsub(q1, q2)` som beräknar `q = q1 - q2`
  3. `q = qmul(q1, q2)` som beräknar `q = q1 * q2`
  4. `q = qdiv(q1, q2)` som beräknar `q` = `q1 / q2` = `q1` $*$ `q2`$^{-1}$ (högerdivision)
  5. `q = qinv(q1)` som beräknar `q` = `q1`$^{-1}$
  6. `a = qabs(q1)` som beräknar `a = |q1|`, där `a` är en `double`
  7. `q = qconj(q1)` som beräknar `q` = $\bar{\texttt{q1}}$
  8. `q = qcons(a, b, c, d)` som konstruerar `q` = `a` + `bi` + `cj` + `dk`, `a`, `b`, `c` och `d` är av typen `double`.
  9. `q = qread()` som läser in `q`.
  10. `void qprint(q)` som skriver ut `q`.

Det saknas rutiner, t.ex. multiplikation av quaternion med ett reellt tal, men dessa rutiner behöver du inte tillhandahålla. Det går också att definiera e^q etc.

Skriv också ett huvudprogram som testar dina rutiner. Här följer *början* på ett skelett:

```
#include <cmath>
#include <iostream>
using namespace std;

// datatype for quaternion
struct quaternion {
  // fill this in!
};

// function prototypes
  quaternion qadd(quaternion, quaternion);
    etc.
  void qprint(quaternion);

int main()
{
  quaternion q, r, s;
  double theta2 = M_PI / 4.0;

  q = qcons(cos(theta2), sin(theta2), 0, 0);
  r = qcons(0, 1, 0, 0);
  qprint(qdiv(qmul(q, r), q));
// or
  s = qdiv(qmul(q, r), q);
  qprint(s);

// check if dqiv(q, q) is equal to one
  qprint(qdiv(q, q));

// more tests ...

  return 0;
}

// computational functions
quaternion qadd(quaternion q1, quaternion q2)
{
 return ...
}

etc.
```