



Grunderna i C++

ARK 385: *Virtuella Verktyg i en Materiell värld*

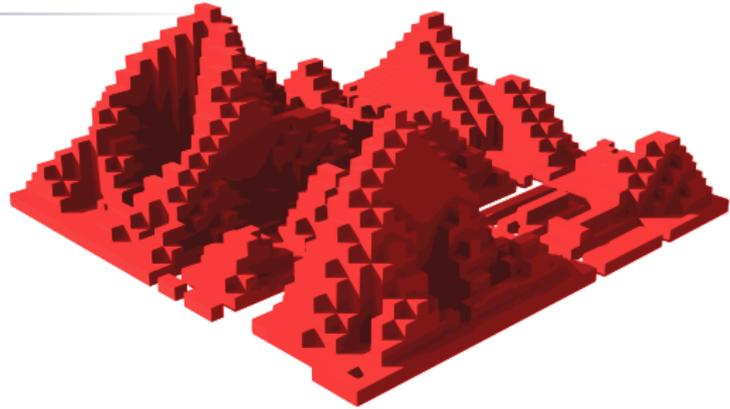
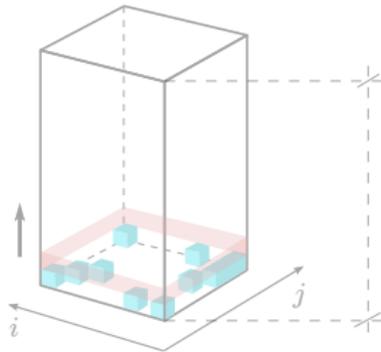
AT Arkitektur & Teknik

Chalmers Tekniska Högskola

2009 - Kursen skapades (3 förel.)

2010 - 6 förel. + 2 projekt

2011 - 8 förel. Helt omarbetade övningar



Skapad av Matz Johansson Bergström.

@ matz.johansson@chalmers.se

Innehållsförteckning I

① Wrap-up

Repetition

Frekventa frågor

Vad vi går igenom idag

② C++ del 5

Avslutande ord

DXF i detalj

LINE

CIRCLE

3DFACE

Processing

Exempel

Fjädersimulering

Övningar

Vad lärde vi oss förra gången?

- Nästlade for-slingor
- 2-dimensionella fält
- Game of Life
- DXF
- OpenGL
- L-System

Frågor från förra gången

Kan man lägga in en radvektor inuti ett tvådimensionellt fält i C++, som i Matlab?

- Kommer jag ta upp i nästa slide.

Hur gör man för att exportera DXF?

-Det kommer vi öva på idag.

Avlusning (fält i fält)

Följande visar en viktig skillnad mellan Matlab och C++:

```
>> x=[2 3 2]; A=[x; 4 5 6; 7 8 9]
A =
     2     3     2
     4     5     6
     7     8     9
```

Radvektor inuti en matris i Matlab.

```
int x[SIZE] = {2, 3, 2};
int A[SIZE][SIZE] = { x, {4, 5, 6}, {7, 8, 9} };
```

Följande går inte att skriva i C++.

Vi får följande meddelande av kompilatorn: **error: braces around scalar initializer for type 'int'**.

Faktum är att **x** och **A** är adresser till positioner i minnet. Om vi skriver ut vad **A** har för värde så får vi ett hexadecimalt tal som säger vart det första elementet är i minnet. Vi är tvungna att gå igenom varje element i **x** och kopiera värdet till vår matris.

Vad lär vi oss idag?

Vi kommer gå igenom lite användbara verktyg till projekten och workshopen.

- DXF-formatet, exempel
- Introduktion till Processing
- Fjädersimulering (projekt 2)

Avslutande ord...

Följande visar en variant på while:

```
do
{
    cout <<"Type the value of n between 1 and 150\n";
    cin >> n;
}
while(n < 1 || n > 150);
```

Initiering av flera variabler samtidigt:

```
int i, j, k;
i = j = k;
```

Exponentialfunktionen finns också: **exp(x)** alltså e^x .

Öka en variabel: $i+=1$ (Chris använder denna ibland) $\Leftrightarrow i=i+1 \Leftrightarrow i++$ (vanlig).

DXF formatet

Vi skall i följande sidor lära oss att skriva till fil i det s.k. DXF-formatet. En DXF-fil är väsentligen en textfil som slutar på DXF och som följer en viss intern syntax.

Dokumentationen kan vara bökig att läsa, den är väldigt generell, se [www](#) därför skall vi gå igenom detaljerna i hur man skapar **linjer**, **cirklar** (egentligen öppna cylindrar) och **Faces**.

Man måste vara noga med att skriva exakt som i dokumentationen och testa import med Rhino eller CAD. Att använda open på DXF-filen går inte. Varje rad skall ha antingen en teckensträng, ett heltal eller flyttal, dvs. varje rad skall sluta med nyrad.

DXF LINE

```

//Början av filen :
Output << "0" << endl << "SECTION" << endl << "2" << endl ...
    << "ENTITIES" << endl;

//Ett linjesegment:
Output << "0" << endl << "LINE" << endl << "8" << endl << ...
    "0" << endl;
Output << "10" << endl << x1 << endl;
Output << "20" << endl << y1 << endl;
Output << "11" << endl << x2 << endl;
Output << "21" << endl << y2 << endl;

//Avslut av filen :
Output << "0" << endl << "ENDSEC" << endl << "0" << "EOF" ...
    << endl;

```

Exempel på hur man skriver till strömmen **Output** med DXF syntax för ett linjesegment.

Där **x1**, **y1**, **x2**, **y2** är flyttal som är definierade innan. Första och sista raden ovan skall alltid stå i en DXF-fil en gång.

DXF LINE

Första raden efter den obligatoriska "SECTION"-raden står att läsa:

```
| Output << "0" << endl << "LINE" << endl << "8" << endl << ...  
|     "0" << endl;
```

Enligt dokumentationen tolkas det som står efter "8" som namn för lagret som linjen skall vara i.

Med hjälp av lager kan man i Rhino välja **Edit** → **Select object** → **by layers...** på så sätt kan man ändra material eller på annat sätt styra sina objekt på ett smidigt sätt.

DXF CIRCLE

Dessa rader skapar "öppna" cylindrar i ett lager "Level 1" med specifikationerna nedan.

```
Output << "0" << endl << "CIRCLE" << endl << "8" << "...
    Level1" << endl;
Output << "39" << endl << height << endl; //höjd

Output << "10" << endl << x << endl;
Output << "20" << endl << y << endl;
Output << "30" << endl << z << endl;

Output << "40" << endl << radius << endl;

//Orientation
Output << "210" << endl << orientation[0] << endl; //x
Output << "220" << endl << orientation[1] << endl; //y
Output << "230" << endl << orientation[2] << endl; //z
```

I detta exemplet har jag definierat **height**, **radius**, **x**, **y** och **z** flyttal. **Orientation** är ett fält av flyttal.

DXF 3DFACE

Vill man rita funktionsytor eller andra 3d-figurer så kan man använda sig av **3DFACES**. Vill man rita kuber måste man skriva varje yta, denna koden skriver man bara en gång och återanvänder.

Exempel på syntax i en nästlad för-slinga med variabler **i**, **j** och 2d-fälten **x**, **y**, **z**:

```
Output << "0" << endl << "3DFACE" << endl << "8" << endl ...
    << "0" << endl;
Output << "10" << endl << i << endl;
Output << "20" << endl << j << endl;
Output << "30" << endl << z[i][j] << endl;

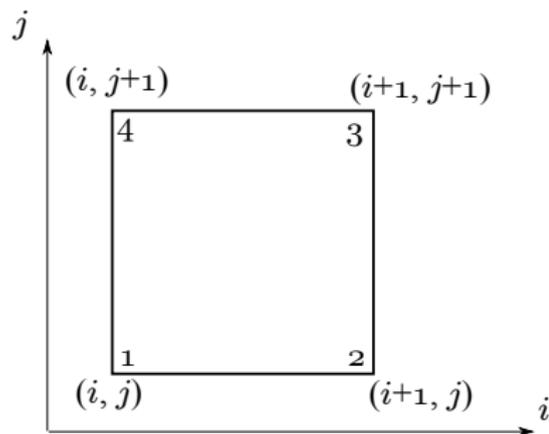
Output << "11" << endl << i+1 << endl;
Output << "21" << endl << j << endl;
Output << "31" << endl << z[i+1][j] << endl;

Output << "12" << endl << i+1 << endl;
Output << "22" << endl << j+1 << endl;
Output << "32" << endl << z[i+1][j+1] << endl;

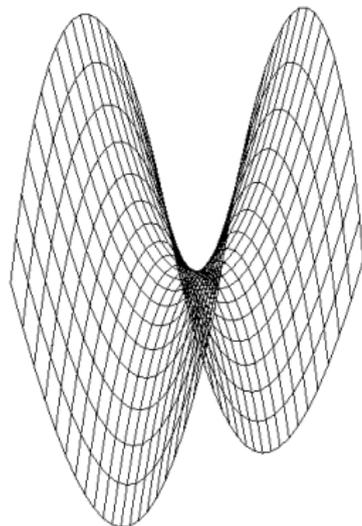
Output << "13" << endl << i << endl;
Output << "23" << endl << j+1 << endl;
Output << "33" << endl << z[i][j+1] << endl;
```

3DFace

Vi vill alltså bygga upp ytor av rektangulära plattor:



DXFs punktordning för 3dface.



Sadelytan $z = x^2 - y^2 |_{x,y \in [-2,2]}$

Vi kan också bygga kuber på samma sätt, detta kommer vi se i projekt 1.

Processing

Processing är ett språk som bygger på Java. Det är väldigt likt C++ med vissa små skillnader, vilka vi kommer gå igenom.

```
| int size      = 100;  
| float [] array = new float [size];
```

- Inga prototyper är nödvändiga.
- Vissa funktioner skall man implementera själv:
setup
draw
- Om man vill styra mus och tangentbord:
keypressed
mouseClicked

Tänka ner Processing och installera, kräver inga administrativa rättigheter [www](#)

Referensblad: [www](#)

Exempel på Processing-kod:

```
void setup()
{
  println("Starting the simulation..."); //utskrift

  size(700, 700, P3D); //storlek på fönstret
  windowSize=700;

  noStroke(); //använd inte penna när man ritar
  frameRate(50); //uppdateringshastighet
  background(0); //svart bakgrund
}
```

Anropet till **noStroke** säger till Processing att inte använda penna när man ritar, dvs. cirklar får inte en rand.

Processing forts.

Tänk på att anropa `size` först i `setup`. I dokumentationen för `size`, se [www](#), säger dem

Again, the `size()` method must be the first line of the code (or first item inside `setup`). Any code that appears before the `size()` command may run more than once, which can lead to confusing results.

- Vi har heltalsdivision om vi dividerar två heltal.
- Typer (`int`, `float`, `double`)
- Funktioner, `abs` (flyttals `abs`, inte riktigt som C++) modulo `%` är
- `main` heter **setup** i Processing

En stor skillnad mellan Processing och C++ är att Processing kan (utan extra tillbehör) returnera fält.

```
float [] update(float [] x)
{
    randomSeed(0);
    float [] res = {random(50), random(50)};
    return res;
}
```

Fjäderssimulering med Eulerintegration

Numeriskt approximera eftersläpande fjäder (ODE):

$$\frac{d^2y}{dt^2} = \underbrace{-ky}_{\text{Hooke's lag}} - \underbrace{d\frac{dy}{dt}}_{\text{Dämpning}}$$

Euler approximerar lösningen till följande differentialekvation:

$$y'(t) = f(t, y(t))$$

Vi vill alltså numeriskt integrera för att hitta $y(t)$. Detta gör Eulers metod genom att approximera funktionen $y(t)$ utmed tangenterna $y'(t)$ i små tidssteg dt . Man inser omedelbart att stora dt för en osnäll funktion y ger stora fel.

Alltså

$$y(t + dt) \approx y(t) + dt y'(t)$$

Fjädern hänger ständigt efter en position **ideal**. Vi inför även fjäderkonstanten samt dämpning.ⁱ

ⁱDessa kan man försiktigt modifiera så den blir dämpad, kritiskt dämpad eller icke dämpad.

Euler forts.

Eftersom Eulers metod för integration bara använder en term för approximation kommer det smyga sig in fel.

```
displacement = position - ideal;
//accelerationen:
aSpring = -kSpring * displace - dSpring * velocity;

//stega i tiden för hastighet och ny position:
velocity += aSpring * dt;
position += velocity * dt;
```

Euler är en dålig approximation av integralen, vi använder ändå metoden för den är enkel att implementera och den fungerar bra för våra "lekexempel".

De två felen (Euler och dt) gör metoden dålig i simuleringar, där använder man högre ordningens polynom (RK4), [www](#)

Övningar

- Ni behöver inte logga in för att hämta ner från kurshemsidan [www](#)
- Öppna **Instruktioner till Övning 5.pdf** (under Övningar/5/)
- Öppna **Referensblad.pdf** (under Extra/)
- Om ni undrar något så kolla i referensbladet, där står allt ni behöver veta.
- Är jag upptagen så kolla på ledtrådarna, annars kan ni skriva upp er på tavlan så tar jag er i tur och ordning. Ni som är snabba får gärna hjälpa övriga.