Fem projektuppgifter

1. Löpande texten – en animation i konsolen

Skriv en C++ Console Application som simulerar en löpande text. Ta som exempel texten **`C++ är kul>`** som ska röra sig horisontellt från konsolfönstrets vänstra kant tills den ”träffar” på ett hinder, t.ex. ett kryss **`X`**. Texten ska börja från vänstra kanten. Krysset ska ligga nära den högra kanten (ca. 70-80 tecken borta). Dessa ögonblicksbilder ska illustrera animationen:

Ledning:

Skriv ut först krysset **`X`** i slutet av en tom rad (fylld med mellanslag). Anteckna hur många mellanslag ni har valt för att placera krysset från konsolens vänstra kant. Gå i samma rad tillbaka till radens början genom att använda escapesekvensen **`\r`** (carriage return). **`\r`** skickar tillbaka markören till början av samma rad, utan att byta rad (till skillnad från **`\n`**). Skriv sedan ut **`C++ är kul>`** som då blir textens initialposition – det som visas i den första ögonblicksbilden ovan. Om ni vill bekanta er mer med **`\r`**:s funktion gör experiment med det i ett annat program.

Rörelsen kan sedan simuleras t.ex. i en **`for`**-loop genom att i varje varv av loopen med ett antal **`\b`** ta bort texten som skrevs ut i förra varvet. Escapesekvensen **`\b`** (backspace) tar bort *ett* tecken till vänster om det aktuella tecknet, precis som tangenten backspace (←). Stega sedan med ett (eller flera) mellanslag, vilket kommer att bestämma rörelsens ”hastighet”. Skriv slutligen om texten **`C++ är kul>`**.

Beräkna antalet varv i **`for`**-loopen genom att ta hänsyn till textens längd och avståndet som kryss **`X`** har från vänstra kanten (som antecknats ovan).

Har ni räknat rätt, kommer rörelsen att stoppas strax före krysset **X**, utan att ta bort det – liknande den tredje ögonblicksbilden ovan.

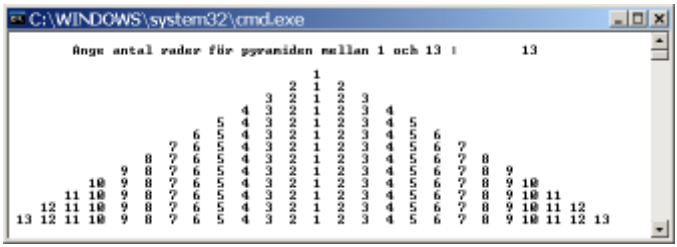
Även om ni gjort allt rätt kommer ni inte ”se” texten att röra sig, eftersom det går så fort, så att ögat inte hinner att se förloppet. Ni måste lägga in en fördröjning, vilket kan göras genom att infoga i loopen t.ex. satsen:

```
Sleep(100);
```

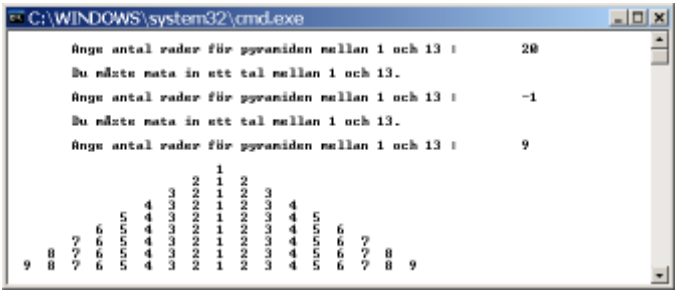
Parameterns enhet är millisekunder. Fördröjningsfunktionen **`Sleep()`** kräver inkluderingen av biblioteket **`windows.h`**.

2. Pyramiden

Slutmålet med detta uppdrag är att utveckla ett program som skriver ut en pyramidliknande figur med tal, som t.ex. ser ut så här:



Programmet ska vara så generellt att det skriver ut talpyramider även om man matar in mindre antal rader. Uppmana användaren att hålla sig till talintervallet [1, 13]. Annars ryms talpyramiden inte i konsolen. Så här kan en körning se ut:



Ledning:

Denna ledning är endast en rekommendation och ska inte förhindra att ni använder egna idéer för att lösa problemet. Det finns andra möjliga tillvägagångssätt. Ni kan använda hela eller också delar av denna ledning för att komma igång.

Man kan *börja* med ett program som ritar en pyramid av *stjärnor* istället för tal:

Strunta till att börja med även på hanteringen av felinmatning av antal rader. Jobba med ett fast antal rader. Du kan lägga till det senare.

Använd en nästlad for-sats med en yttre loop och tre inre loopar:

- En för de tomma platserna i pyramiden (mellanslagen),
- En för stjärnorna i pyramidens högra halvan (räknat från den vertikala mittlinjen (symmetriaxeln),
- En för stjärnorna i pyramidens vänstra halvan.

Räkna med att ni måste använda i de inre looparna den yttre loopens räknare och slutvärde. T.ex. kan villkoret i den första inre loop som ritar de tomma platserna, se ut så här:

```
column <= numberOfRows - row;
```

Här är **`column`** den inre loopens, **`row`** den yttre loopens räknare och **`numberOfRows`** hela pyramidens antal rader, t.ex. 13. Då kan den första inre loopen skriva ut tre mellanslag i varje varv. I de två andra inre looparna kan två mellanslag och en **`*`** skrivas ut.

3. Labyrinten

Visst är det roligt att med våra C++ kunskaper hittills kunna skriva ett program som ritar en labyrintartad figur i konsolen som kan se ut så här:

Detta är förstås inte någon riktig labyrint. För en sådan skulle kräva mycket mer. En riktig labyrint med in- och utgång osv. skulle kunna ingå i ett spelprojekt med grafiska finesser, färger osv. En sådan avancerad figur kan inte ritas i konsolen.

Vår uppgift går snarare ut på att i *text mode* ”rita” en *labyrintartad figur* som är slumpmässigt ihopsatt av mellanslaget och ett antal tecken som vi kallar för *dubbla linjegrafiktecken (LGT)*. I figuren ovan är dessa tecken slumpade i en 2D utskrift, en slags tabell eller matris med 50 rader och 20 kolumner. I koden åstadkommer man detta med en nästlad **`for`**-loop, där den yttre loopen skriver ut raderna och den inre loopen kolumnerna, se avsn. ***6.8 Nästlade for-satser***, sid 155. Alla tecken i figuren ovan är slumpvist valda bland de dubbla LGT och mellanslaget. Därför borde varje körning av programmet generera en lite annorlunda labyrintartad figur.

Du kan gärna försöka med en egen algoritm att skriva ett C++ program som ritar en sådan figur. Men i instruktionerna som följer nedan har du i alla fall ett förslag till en enkel algoritm (***Steg 1-5***) som fungerar.

Algoritmen

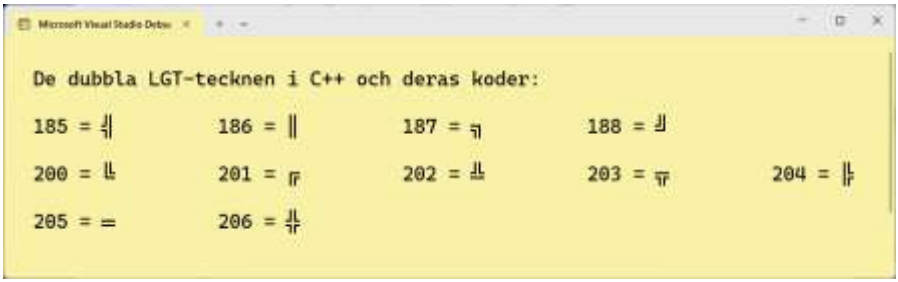
Steg 1 Bekanta dig med hantering av tecken i C++ inkl. explicit typkonvertering (sid 118), genom att experimentera med programmet **`Int2char`** som behandlades tidigare (sid 119):

```
// Int2char.cpp
// Ger tecknet till en inmatad kod
// Representation av tecken med heltalskoder
#include <iostream>
using namespace std;

int main()
{
   int code;
   cout << "\nMata in ett heltal: ";
   cin >> code;
   cout << "\nDet inmatade heltalet är " << code <<
        " och är koden till tecknet " << (char) code << "\n";
}
```

Experimentera med **`Int2char`** genom att mata in koderna **`185-188`** och **`200-206`**.

Steg 2 För få en översikt över alla elva dubbla LGT samt deras koder i C++ skriv ett program som producerar följande utskrift:



Dessa tecken finns i den utvidgade delen av teckentabellen (utöver standard ASCII) och används för att rita raka linjer, ramar, tabeller, skisser osv i en textbaserad miljö *(text mode)*. De kan användas tillsammans med mellanslaget för att rita labyrinten. Jämför koderna även med utskriften till programmet **`AsciiFor`** (sid 153).

Steg 3 Repetera hantering av slumptal i avsn. ***4.9 Hantering av slumptal*** (sid 93), speciellt om ***Slumptal inom ett intervall*** (sid 94).

Steg 4 Skriv ett program som med hjälp av C++:s slumpgenerator och en nästlad **`for`**-sats ritar labyrinten, en slumpmässigt ihopsatt figur bestående av de dubbla LGT samt mellanslaget.

Steg 4 i detalj

Här följer i själva verket hur ***Steg 4*** kan realiseras:

Deklarera en teckenvariabel **`letter`** och en heltalsvariabel **`randNo`**. Initiera **`letter`** till mellanslaget, dvs **`' '`**. Låt **`randNo`** anta slumpvärden mellan **`0`** och **`11`** med satsen:

```
randNo = rand() % 12;
```

Fortsätt med följande **`if`**-satser:

Om **`randNo`** blir **`0`**	ska **`letter`** tilldelas	1:a LGTs teckenkod.
Om **`randNo`** blir **`1`**	ska **`letter`** tilldelas	2:a LGTs teckenkod.
Om **`randNo`** blir **`2`**	ska **`letter`** tilldelas	3:e LGTs teckenkod.
.	.	
.	.	
.	.	
Om **`randNo`** blir **`9`**	ska **`letter`** tilldelas	10:e LGTs teckenkod.
Om **`randNo`** blir **`10`**	ska **`letter`** tilldelas	11:e LGTs teckenkod.
Om **`randNo`** blir **`11`**	ska **`letter`** tilldelas	mellanslaget, dvs **`' '`**.

Numreringen av LGT-teckenkoderna avser den ordning som är föregiven i teckentabellen, se utskriften på förra sidan. I övrigt fungerar vilken numrering som helst.

Observera att LGT-teckenkoderna är av typ **`int`**, medan variabeln **`letter`** är av typ **`char`**. Vid tilldelningen konverteras **`int`** automatiskt till **`char`**. Vid utskriften med **`cout`** skrivs ut tecknet, inte koden. Mellanslaget (**`' '`**) däremot är från början av typ **`char`**, så att vi inte behöver bry oss om koden. Se upp att **`''`** inte är mellanslaget utan den tomma strängen och ger kompileringsfel.

Skriv ut **`letter`**, så att du får ETT tecken, antingen ett LGT eller mellanslaget. Testa även om du vid varje körning får *olika* LGT eller mellanslaget.

Först när allt detta fungerar låt tilldelningen av variabeln **`randNo`** samt de 12 **`if`**-satserna ovan ingå i en enkel *loop*, t.ex. en **`for`**-sats, för att rita labyrinten.

Ersätt den enkla loopen med en nästlad **`for`**-sats och lägg in ett radbyte mellan den inre och yttre slingan för att kunna styra labyrintens storlek (höjd och bredd) vid varje körning. Låt användaren ange höjd och bredd.

Algoritmen ovan är enkelt genomförbar, men inte den mest eleganta lösningen. Har du klarat av den kan du gärna gå vidare till följande:

Extrauppgift (frivilligt)

Ersätt de 12 **`if`**-satserna ovan med en annan konstruktion: Samla alla dina LGT-koder och mellanslaget i en *array* och låt slumpen ta ut tecken ur denna array. För utskriften kan du fortsätta med att använda nästlad **`for`**-sats. Det blir en kortare och elegantare kod.

4. Kalkylatorn

I denna uppgift ska ett program **`Calculator`** skapas som stödjer följande funktionaliteter: addition, subtraktion, multiplikation, division och potentiering samt att kunna ange det största och minsta av två inmatade tal.

Kalkylatorn ska vara igång kontinuerligt tills användaren väljer att stänga av den, vilket innebär att ni måste lägga in en loop. De olika räkneoperationerna ska definieras i separata funktioner och anropas i **`main()`**.

Följande funktioner ska definieras i programmet **`Calculator`**:

```
double add(double operand1, double operand2)
{
     // Additon av operand1 och operand2
}


double sub(double operand1, double operand2)
{
     // Subtraktion: operand1 - operand2
     // Även subtraktion av negativa tal
}


double mult(double operand1, double operand2)
{
     // Multiplikation av operand1 med operand2
}


double div(double operand1, double operand2)
{
     // operand1 / operand2
     // Division med 0 ska förhindras
     // Felmeddelande vid inmatning av 0 för opernad2
}


double pow(double operand1, double operand2)
{
// Beräkning av potens: operand1 upphöjt till operand2

}

double max(double operand1, double operand2)
{
     // Returnera det större värdet av operand1 och operand2
}

double min(double operand1, double operand2)
{
     // Returnera det mindre värdet av operand1 och operand2
}
```

Programmet skall exekvera kontinuerligt tills användaren väljer att avsluta körningen. För att åstadkomma detta kan ni exempelvis använda en **`do`**-sats. Kalkylatorn kan avslutas genom att användaren matar in t.ex. tecknet **`q`** (quit) istället för en operator.

Placera först, när ni börjar koda, all kod för programmet **`Calculator`** i en fil. Flytta sedan alla ovannämnda funktioner till en annan fil. Bibehåll endast **`main()`** som läser in data, anropar funktionerna och skriver ut resultat.

Frivilligt! I slutet, när allt fungerar, försök att lägga in kod som hanterar ev. felaktiga inmatningar. Detta borde inkludera både inmatning av operander, men även av operatorer.

Se till att du skriver ut meningsfulla felmeddelanden, så att användaren får möjligheten att rätta till sin felaktiga inmatning.

5. Time

Programmet **`Hour2Sec`** (sid 85) omvandlar timmar, minuter och sekunder till totalsekunder. Programmet **`Sec2Hour`** nedan löser det omvända problemet: att omvandla ett givet antal totalsekunder till timmar, minuter och sekunder.

```
// Sec2Hour.cpp
// Omvandlar antal sekunder till timmar, minuter & sek.

#include <iostream>
using namespace std;

int main()
{
   int tim, min, sek, totalsek;

   /* I n m a t n i n g */
   cout << "\nAnge tid i sekunder: ";


   cin  >> totalsek;

   /* B e a r b e t n i n g */
   tim =   totalsek / 3600;
   min =  (totalsek % 3600) / 60;
   sek = ((totalsek % 3600) % 60) % 60;

   /* U t m a t n i n g */
   cout << '\n' << totalsek << " totalsekunder = "
        << tim
        << " timmar, " << min << " minuter, " << sek
        << " sekunder.\n\n";
}
```

En körning av programmet **`Sec2Hour`** ger t.ex. följande dialog:

```
Ange tid i sekunder: 15910

15910 totalsekunder = 4 timmar, 25 minuter, 10 sekunder.
```

Modularisera programmet **Sec2Hour** genom att flytta bearbetnings- och utmatnigsdelen till en **void**-funktion. Dvs skriv ett program som läser in tiden i ett antal sekunder, anropar **void**-funktionen som omvandlar tiden till antal timmar, minuter samt resterande sekunder och skriver ut resultaten.

Använd för omvandlingen den algoritm som är implementerad i programmet **Sec2Hour**. Varför kan man inte här använda en funktion med returvärde?