Innehållsförteckning

4 Programflödeskontroll

Detta avsnitt behandlar de tre grundstenarna inom procedurell programmeringen är sekvens, upprepning och val.

- **Sekvens**
  En sekvens är en rad av operationer som görs en efter en.
- **Upprepning** (**Iteration**)
  Vid upprepning görs en eller flera satser ett antal gånger.
  I C++ finns 3 konstruktioner för upprepning: **while, do – while** och **for.**
- **Val** (**Selection**)
  Vid val väljs en av flera möjliga ”vägar” i programkoden med hjälp av konstruktionerna **if** och **switch**

I detta avsnitt tar jag också upp hur man **slumpar** tal samt variablers **synlighet** och **livslängd**,

4.1 Val

4.1.1 if-satsen

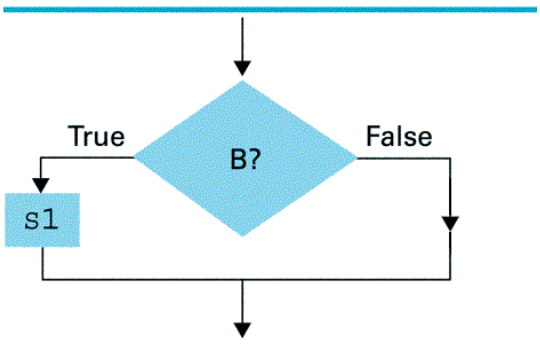
Med if-satsen väljer man om en eller flera satser ska utföras eller inte. Ett första exempel:

```
// select_010
// if-satsen - avgör om ett tal är positivt
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>    // cout
using namespace std;
//--------------------------------------------------------------------------
int main()
{

  int num  ;

  cout << " Input a number: ";
  cin >> num;
  if (num > 0)
    cout << " the number is positive" << endl << endl;

  return 0;
}
```

Kommentarer:

- Syntax **if** (*villkor*)
  sats;
- villkoret är sant eller falskt
- En sats utförs. Vill man att flera satser ska utföras får man sätta parenteser:

```
if (villkor)
{
  sats1;
  sats2;
}
```



4.1.2 if-else-satsen

Med denna sats kan välja att köra antingen sats1 eller sats2. Exempel med pseudokod:

OM villkor är sant
  sats1
ANNARS
  sats2

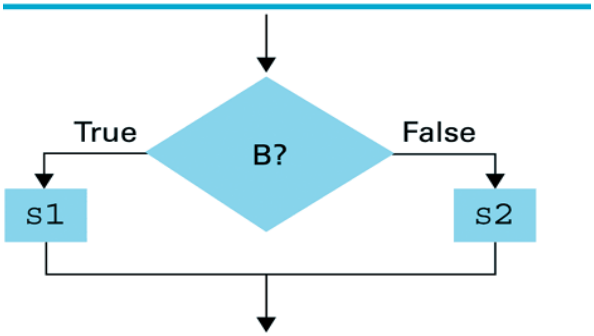
```
// select_020
// if - else - satsen  - avgör om ett tal är positivt eller negativt
// Per Ekeroot 2013-09-01
// Ver 9
//-----------------------------------------------------------------------------
#include <iostream>   // cout
using namespace std;
//-----------------------------------------------------------------------------
int main()
{
  int num;

  cout << " Input a number: ";
  cin >> num;
  if (num > 0)
    cout << " The number is positive" ;
  else
    cout << " The number is negative";
  cout << endl << endl;

           return 0;
}
```

Kommentarer:

- **Syntax**:

1) En sats i respektive fall:

```
if (villkor)
  sats_om_sant;
else
  sats_om_falskt;
```



2) Flera satser i respektive fall:

```
if (villkor)
{
  sats1_om_sant;
  sats2_om_sant;
}
else
{
  sats1_om_falskt;
  sats2_om_falskt;
}
```

Det går att nästla flera if-satser i varandra. Här visas ett exempel med kontroll av om ett tal är större, mindre eller lika med noll, d.v.s. tre alternativ:

```
// select_030
// if - else - if - satsen. Avgör om ett num är positivt, negativt eller noll
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>   // cout
using namespace std;
//--------------------------------------------------------------------------
int main()
{
  int num;

  cout << " Input a number: ";
  cin >> num;

  if (num > 0)
    cout << " The number is positive" ;
  else if( num < 0 )
    cout << " The number is negative";
  else
    cout << " The number = 0";
  cout << endl << endl;

           return 0;
}
```

Kommentarer:

- Här kollar man tre alternativ: mindre än, större än och lika med
- Syntax

```
if (villkor1)
  sats_om_villkor1_sant;
else if (villkor2)
  sats_om_villkor2_sant;    | Om villkor1 är falskt
else                        | görs dessa rader
  sats_om_villkor2_falskt;  |
```

- För att få flera satser i respektive fall sätts parenteser {…} pss tidigare.
- Det går att fortsätta nästlingen av if-else-satserna.

4.1.3 switch – satsen

if-else-if-else-if-satsen blir klumpig om för många villkor (steg) ska kontrolleras. Som alternativ finns då switch-satsen.

```
// select_040
// switch - satsen
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>
using namespace std;
//--------------------------------------------------------------------------
int main()
{
  // Deklaration av variabel och inmatning av tal
  int num;
  cout << " Input an integer: ";
  cin >> num;

  // Testar talet och skriver resultatet
  cout << " The number is ";
  switch (num)
  {
     case 1 : cout << "one";
     break;
     case 2 : cout << "two";
     break;
     case 3 : cout << "three";
     break;
     case 4 : cout << "four";
     break;
     default : cout << "not in my list!";
  }
           cout << endl << endl;

           return 0;
}
```

Kommentarer:

- Syntax:

```
switch(testvariabel)  // int eller char
{
  case etikett1 : sats1;
  case etikett2 : sats2;
  case etikett3 : sats3;
  case etikett4 : sats4;
  default: sats
}
```

- Testvariabeln måste vara av heltalstyp (int eller char).
- Etikett1- 4 är möjliga utfall av testvariabeln.
- Programmet hoppar till den rad vars etikett motsvarar testvariabeln.
- Observera att även resterande satser i switch-uttrycket körs
- Om man vill att resterande satser inte ska köras sätter man in ett **break** som gör att programmet fortsätter direkt efter switch-satsen.
- Det går inte att testa logiska uttryck, t e x a>b.
- Om testvariabeln inte stämmer in på någon av etiketterna 1- 4, körs satsen vid default
- Default kan uteslutas

4.1.4 switch – satsen med char

Ett exempel till som visar att man kan använda variabler av datatypen char som testvariabler. Anledning till att det går att använda char-variabler är att char är en heltalsdatatyp.

```
// select_050
// switch - satsen med tecken (char)
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>   // cout
using namespace std;
//--------------------------------------------------------------------------
int main()
{
  // Deklarera en teckenvariabel
  char ch;

  // Mata in ett tecken
  cout << " Input  character: ";
  cin >> ch;

  // Skriv olika strängar beroende på inmatning
  switch (ch)
  {
     case 'a' :
     case 'A' : cout << " Adam";
     break;
     case 'b' :
     case 'B' : cout << " Bertil";
     break;
     case 'c' :
     case 'C' : cout << " Cesar";
     break;
     case 'd' :
     case 'D' :cout << " David";
     break;
     default : cout << " The character is not in my list!!";
  }

  cout << endl << endl;
  return 0;
}
```

Kommentarer:

- om man t ex trycker **a** så körs satsen cout << ”Adam”; som står vid etiketten ’A’ eftersom break är placerat efter denna sats.
- Observera att char är ”av heltalstyp”

4.2 Upprepning

Upprepning eller iteration utförs med konstruktionerna **while**, **do** – **while** eller **for**-satserna.

4.2.1 while – loopen

I första while- exempel ska vi skriva ett program som löser uppgiften:

Summera talen 1+2+3+4+… så länge som summan är mindre än 100.

```
// repeat_010
// while-loopen. Ett första exempel
// Per Ekeroot 2013-09-01
// Ver 9
//-----------------------------------------------------------------------------
#include <iostream>   // cout
using namespace std;
//-----------------------------------------------------------------------------
int main()
{
  //Deklarera två variabler
  int nr = 0, sum = 0;

  // Summera så länge som summan < 100
  while ( sum < 100 )
  {
    nr++;
    sum += nr;                // sum = sum + nr;
  }

  // Skriv resultatet
  cout << " Sum of the numbers 1 - " << nr << " is " << sum << endl << endl;

  return 0;
}
```

Kommentarer:

- sum+ = nr så länge som sum < 100.
- nr räknas upp ”manuellt” nr++
- nr++ är det samma som nr = nr +1;

Syntax:

```
while(villkor)
  sats;
```

- *sats* utförs så länge som *villkor* är **sant** (fortsättningsvillkor).
- använd parenteser { } om mer än en sats ska vara med i loopen.
- OBS! Villkoret testas först, sedan utförs eventuellt satsen.

Ofta ser det ut så här:

```
int counter = 1;  // ge startvärde
while( counter < avbrottsVärde)
{
  sats(er);
  counter ++;
}
```

I nästa exempel ska vi konstruera en fördröjning med hjälp av en while-loop.

```
// repeat_020
// while-loopen
// Tidsfördröjning med while()
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>   // cout
#include <ctime>       // clock()
using namespace std;
//--------------------------------------------------------------------------
int main()
{
  // Mata in data
  cout << " Input time delay [seconds]: " ;

           int secs;
  cin >> secs;
  cout << " START \a" << endl;

  // Gör beräkningar
  clock_t delay = secs * CLOCKS_PER_SEC; // Konvertera till klocktid
  clock_t start = clock();               // Läs av systemklockan

  while( clock() - start < delay)
     ;  // Gör INGENTING tills tiden har gått

  cout << " STOP \a" << endl << endl;

  return 0;
}
```

Kommentarer:

- Inkludera standardbiblioteket time
- Systemtiden ges **inte** i sekunder utan i s.k. clocks. Antalet clocks per sekund ges av konstanten CLOCKS_PER_SEC
- Funktionen clock() läser av datorns klocka och returnerar ett tal på formatet clock_t.
- I while-satsens villkor läses klockan för varje varv den snurrar och den aktuella tiden jämförs med starttiden.
- När denna skillnad är mindre än den önskade fördröjningen görs INGENTING!!
- När denna skillnad är större än den önskade, avbryts WHILE-loopen och programmet fortsätter.
- \a = bell, det piper i datorns interna högtalare (\a är en Escape-sekvens, se avsnitt 2)

4.2.2 do-while-loopen

while-loopen testar villkoret **innan** satsen utförs. Om man vill att satsen ska köras först och villkoret ska testas sedan, använder man **do – while** – konstruktionen.

Exempel: Gissa det gömda talet!

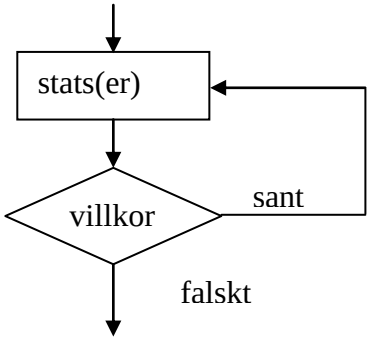
```
// repeat_030
// do - while-loopen Ett första försök
// Per Ekeroot 2013-09-01
// Ver 9
//-----------------------------------------------------------------------------
#include <iostream>   // cout
using namespace std;
//-----------------------------------------------------------------------------
int main()
{
  // Deklarera en heltalsvariabel och en boolsk variabel
  int n;
  bool ok = false;

  cout << " Try to guess the hidden number (1-10)!" << endl;
  do
  {
    cout << "I guess: ";
    cin >> n;
    ok = n==7;
    if(!ok)
      cout << "WRONG! Try again!" << endl << endl;
  }while (!ok); // Repetera så länge n INTE är 7

  cout << endl << endl << "HIT!! The hidden number is 7!" << endl << endl;

  return 0;
}
```

Kommentarer:

- Inmatningssatsen cin >> n; upprepas så länge 7 INTE matas in
- OBS! cin >> körs ALLTID EN gång!
- ! betyder **inte**
- I satsen `ok = n==7;` utförs `n==7` först eftersom operatorn == har högre prioritet än =. Om n är lika 7 blir `hit` **true** annars **false**.

Syntax:

```
do
{
  sats(er);
}while(villkor);
```



Kommentarer:

- sats(er) utförs så länge **villkor** är sant.
- sats(er) körs **minst en** gång

Nästa exempel är en variant på det förra. Här får användaren möjlighet att köra programmet mer än en gång och avbryta enligt önskemål. Dessutom kontrolleras vilka tecken som användas, enbart y, n, Y, N och ENTER är tillåtna vid svar på frågan *Köra programmet en gång till? (y/n)[Y]:*". Om man gissar fel så skrivs meddelandet: *x is WRONG! Try again!* där x är talet man gissat på.

```
// repeat_040
// Per Ekeroot 2013-09-01
// Ver 9
/-------------------------------------------------------------------------
#include <iostream>     // cout
using namespace std;
//-------------------------------------------------------------------------
int main()
{
  // Deklarera variabler
  int n;
  char ch;
  // Programmets huvudloop ------------------------------------------------
  do
  {
    cout << " Try to guess the hidden number (1-10)!" << endl;

    // Gissa-rätt-loopen ---------------------------------------------------
    bool ok = false;
    do
    {
      cout << endl << "I guess: ";
      cin >> n;
      ok = n==7;
      if(!ok)
        cout << endl << n <<" is WRONG! Try again!" << endl;
    }while(!ok);
    //-----------------------------------------------------------------------

    cout << " HIT!! 7 is the hidden number!"<< endl << endl;

    // Fråga användaren
    cout << "Run again? (y/n): " << endl;

    // Tillåtna-tangenter-loopen -------------------------------------------
    do
    {
      cin >> ch;
      ch = toupper(ch);          // Gör om till versal
    }while( !(ch == 'Y' || ch == 'N'));
                                 // Endast y, Y, n, eller N är tillåtna tangenter
    // -------------------------------------------------------------------

  }while(ch == 'Y');
  // Slut på  programmets huvudloop ---------------------------------------

 return 0;
}
```

Kommentarer:

- Först deklareras två variabler, en int och en char (ett tecken).
- två do-whileloopar ligger (nästlade) inuti en yttre huvudloop.
- getch() tar emot inmatning från tangentbordet utan bekräftelse av ENTER
- toupper() konverterar gemener till versaler. Funktionen konverterar a-z, dock **inte å, ä, ö**!
- || betyder logiskt ELLER (OR)

4.2.3 for – loopen

for-loopen är ett alternativ om man vill upprepa en sats flera gånger och man dessutom vet hur många gånger stasen ska utföras

4.2.3.1 Upprepa en sats 5 gånger

for-satsen (for-loopen) upprepar en sats ett visst antal gånger.
Exempel: Skriv texten ”for-loopen skriver samma text många gånger” 5 gånger på skärmen:

```
// repeat_050
// for-loopen
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>     // cout
using namespace std;

//--------------------------------------------------------------------------
int main()
{

  for( int i=0; i < 5; i++)
    cout << "The for loop prints the text several times" << endl;

  return 0;
}
```

Syntax:

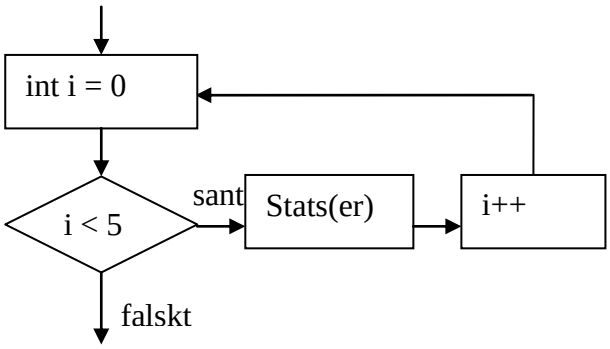
```
  for (startvärde; villkor; uppdatera)
    sats;
```

Kommentar:

- Villkoret är ett *fortsättnings*villkor.

Om mer än en sats ska upprepas används parenteser:

```
for (startvärde; villkor; uppdatera)
{
  sats1;
  sats2;
}
```



4.2.3.2 Använd loopräknaren i loopen

```
// repeat_060
// For-loopen: använd loop-räknaren vid utskrift
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>    // cout
using namespace std;
//--------------------------------------------------------------------------
int main()
{
   for(int i=0 ;i<6 ; i++)
     cout << "The for loop prints row nr " << i+1 << endl;

   return 0;
}
```

Kommentar:

- Här används räknarvariabeln (loop-variabeln) i utskriftssatsen.
- loopräknaren (i) har sitt scope (giltighetsområde) inom for – satsen.

4.2.3.3 Ytterligare exempel på for-loop-konstruktioner:

```
//"Normalfallet" = uppåträkning
for( int i=0; i < 100; i++)
  sats;

// Nedåträkning
for( i=100; i > 0 ; i--)
  sats;

// Speciella villkor för fortsättning och uppräkning
for( i=10; i < x*y; i=2*i)
  sats;

// Räknarvariabeln som flyttal
for( float x=0.0 ; x < 5.0; x=x+0.1)   // x += 0.1
  sats;
```

4.2.3.4 Ett exempel där loopen har mer än en sats

Exempel: Låt användaren mata in två tal som multipliceras. Skriv ut resultatet. Upprepa detta tre gånger.

```
// repeat_070
// For-loopen: använd loop med flera satser
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>    // cout, cin
using namespace std;
//--------------------------------------------------------------------------
int main()
{
   // Deklarera två flyttal
   float num1,num2;

   // Upprepa multiplikation 3 gånger
   for(int i=0; i<3; i++)
   {
     // Rubrik
     cout << " Multiplication nr " << i+1 << endl;

     // Inmatning
     cout << " Input first number : ";
     cin >> num1;
     cout << " Input second number: ";
     cin >> num2;

     // Beräkning och utmatning
     cout << " "<< num1 <<" * "<<num2 << " = " << num1*num2 << endl<<endl;
   }

   return 0;
}
```

Kommentarer:

- Raderna som ska upprepas placeras inom { }

```
for(int i=0 ;i<3 ; i++)
{
  . . .
}
```

- Loop-variabeln (i) har sitt scope mellan parenteserna.

4.2.4 Dubbla for-loopar

Med dubbla for-loopar kan man skriva en tabell.
Exempel: Skriv en tabell med 12 kolumner och 10 rader. Beräkna och skriv radNr*kolumnNr på följande sätt:

Rad/kol	1	2	3	4	5
1	1	2	3	4	…
2	2	4	6	8	…
3	3	6	9	12	…

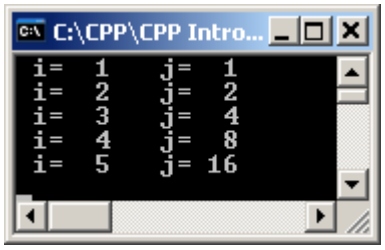
```
// repeat_080
// for-loopen: gör en tabell med dubbla loopar.
// Per Ekeroot 2013-09-01
// Ver 9
//-------------------------------------------------------------------------------
#include <iostream>   // cout
#include <iomanip>    // setw()
using namespace std;
//-------------------------------------------------------------------------------
int main()
{
   for(int row=1;  row <= 10 ; row++)  // Loop för antalet rader
   {
     for( int col=1; col <= 12; col++) // Loop för antalet kolumner
       cout << setw(5)<< row*col;
     cout << endl;                     // Byt rad
   }
   return 0;
}
```

Kommentarer:

- De två looparna skapar en tabell med 10 rader och 12 kolumner
- setw(5) gör plats för 5 tecken som skrivs högerjusterade. Inkludera <**iomanip**>.
- setw() måste upprepas för varje cout!

4.2.4.1 Använd flera variabler i for-loop-huvudet

Detta är ett exempel som visar hur man kan göra en ”smartlösning”.
Exempel: Skriv en for-loop som skriver ut följande tabell




```
// repeat_090
// For-loopen: använd flera variabler i en for-loop
// Per Ekeroot 2013-09-01
// Ver 9
//-----------------------------------------------------------------------------
#include <iostream>   // cout
#include <iomanip>
using namespace std;
//-----------------------------------------------------------------------------
int main()
{
   for( int i = 1, j = 1 ; j < 5 * i ; i++, j = 2*j)
     cout << " i= " << setw(2)<< i << "   j= " << setw(2) << j << endl;

  return 0;
}
return 0;
}
```

Kommentarer:

- **int** gäller för både i och j
- Komma (,) separerar flera argument (komma-operatorn)
- Det **kan** bara finnas och det finns bara **ett** fortsättningsvilkor
- i och j räknas upp på olika sätt
- Det kan vara svårt att inse hur detta kompakta skrivsätt fungerar och jag rekommenderar normalt inte sådana här "smartlösningar".

4.3 Slumpning

Slumpning av tal kan göras på många olika sätt. I den nya standarden ingår nya sätt att göra slumpning. Jag har valt att visa hur man slumpar heltal i angivet intervall med en av de nya metoderna i C++11.
I biblioteket random slumpar man tal genom att använda *generators* och *distributions*.

Generators: Objekt som genererar tal som är fördelade enligt vald distribution.
Distributions: Objekt som transformerar talserier som är genererade av en generator till en serie som följer en speciell distribution. En distribution kan t.ex vara uniform, normal eller binomial.

Exempel: Slumpa ett tal i intervallet [1,100] (inklusive gränserna). Sannolikheten för de olika talen är lika stor (uniform)

```
uniform_int_distribution<int> random1(1,100);
int num = random1(generator);
```

Vid anrop av generatorn *random1* genereras talen från en matematisk funktion som ser till att varje tal i intervallet har lika stor sannolikhet. *random1* är ett namn på generatorn som programmeraren själv anger, ungefär som ett variabelnamn. Vid upprepade anrop av *random1* skapas en serie av tal.

Om dessa upprepade anrop görs från ett program som körs flera gånger, kommer samma serie av tal att genereras. Man kan i början av programmet anropa ett frö som anger var i talserien det första talet ska hämtas. För att "slumpa" ett värde på fröet kan man skriva så här:

```
  default_random_engine generator(static_cast<unsigned>(time(0));
```

Här använder man standardfunktionen time (finns i ctime) som returnerar ett stort heltal (= antalet sekunder som gått sedan ett visst datum), detta tal används som frö. Generator() vill ha en parameter som är unsigned, time() returnerar inte ett unsigned tal, därför får man typomvandla returvärdet från time()!

Ett exempel som slumpar tal i intervallen [1,100] och [-50,50] (inklusive gränserna!):

```
// func_020
// Visar hur man använder slumpfunktioner
// Per Ekeroot 2013-09-01
// Ver 9
//-----------------------------------------------------------------------------
#include <iostream>  // cout, cin
#include <iomanip>   // setw()
#include <ctime>     // time()
#include <random>    // random generator,distribution
using namespace std;
//-----------------------------------------------------------------------------
int main()
{
  // Initiera slumpning
  default_random_engine generator(static_cast<unsigned>(time(0)); // Ser till
att raden av slumptal inte upprepas

  // Slumpa 20 st heltal i intervallet 1 till 100
  // Skriv sedan ut med 5 tal på varje rad
  cout <<" Generate 20 integers between 1 and 100" << endl;
```

```
// Ange att heltal ska slumpas jämnt över intervallet [1,100]
  uniform_int_distribution<int> random1(1,100);

  for (int i=0; i<20; i++)
  {
    cout << setw(6) << random1(generator); // Slumpa tal som skrivs ut
    if((i+1)%5==0)
      cout << endl;
  }
  cout << endl;

  // Slumpa 20 st heltal i intervallet -50 till 50
  // Skriv ut med 5 tal på varje rad
  cout << endl <<" Generate 20 integers between -50 and 50" << endl;

// Ange att heltal ska slumpas jämnt över intervallet [-50,50]
  uniform_int_distribution<int> random2(-50,50);
  for (int i=0; i<20; i++)
  {
    cout << setw(6) << random2(generator);
    if((i+1)%5==0)
      cout << endl;
  };
  cout << endl << endl;
  return 0;
}
```

Kommentarer:

- `default_random_engine generator(time(0));` ska enbart köras **en** gång!! Så placera den i början av programmet, absolut inte inne i en loop eller i en funktion.
- `random1` slumpar tal i intervallet [1,100]
- `random2` slumpar tal i intervallet [-50,50]

4.4 Variablers räckvidd

En variabel har ett visst begränsat område i ett program där den lever, den har en viss räckvidd. På engelska heter detta **scope**.

Huvudregeln är att en variabels räckvidd (scope) är inom de parenteser där den är deklarerad

```
// synlighetsområde
{
  int tal;
}
```

Exempelprogram:

```
// progstruct_050
// Variablers synlighet och livslängd
// Per Ekeroot 2013-09-01
// Ver 9
//--------------------------------------------------------------------------
#include <iostream>
using namespace std;
//--------------------------------------------------------------------------
int main()
{
  {
    int num = 17;
    cout << " Number = " << num << endl;
  }
  //cout << num << endl;  // Variablen 'num' ej synlig (åtkomlig) här!

  cout << endl << endl;
  return 0;
}
```

Kommentarer:

- Ta bort // på raderna med // ” … ej synlig här!” och se vad som händer