3 Styrning av programflöde

Ett program består av ett antal satser. När programmet körs exekveras satserna i den ordning som de står.

Ex:
```
{
  sats1;
  sats2;
  sats3;
}
```

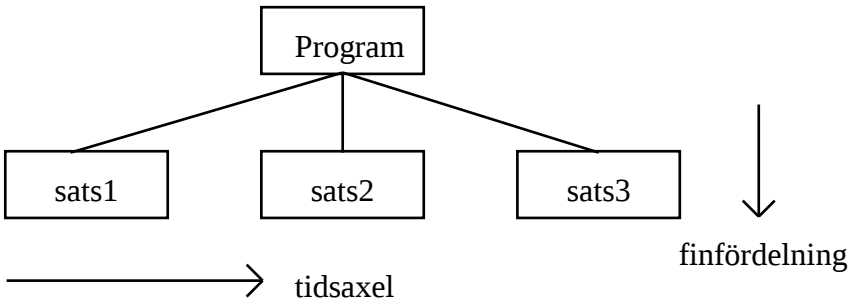
Här exekveras satserna enligt först sats1 sedan sats2 och sist sats3.

Ofta måste man i programmet kunna förändra ovanstående programflöde för att få en korrekt och önskad bearbetning av data. Man ska kanske bara utföra sats2, om något villkor är sant eller man vill upprepa sats3 fem gånger.

I ett programspråk måste det finnas möjligheter att styra det normala eller sekventiella programflödet genom att välja (selektera) eller upprepa (iterera) vissa satser. De styrmekanismer som behövs för att bryta *sekvenserna* är *selektioner* och *iterationer*. Med hjälp av dessa styrmöjligheter kan man lösa alla bearbetningar av data som kan behövas.

När man konstruerar sina program eller databearbetningar är det ofta lämpligt att först tänka ut en plan hur bearbetningen ska göras. Istället för att då använda ett programmeringsspråk kan man rita ett *strukturdiagram*, eller skriva så kallad *pseudokod*, även kallad *halvkod*. I pseudokod beskriver man sin bearbetning på vanlig svenska, men man kan också blanda in konstruktioner från ett programmeringsspråk.

Ex: Ett strukturdiagram som visar ovanstående sekvens.



Ex: Pseudokod för ovanstående sekvens.

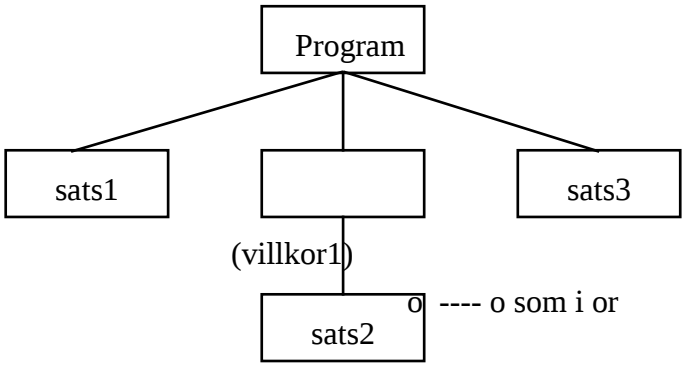
```
Program
  sats1
  sats2
  sats3
```

3.1 Selektion

Med styrmekanismen *selektion* kan man välja att utföra satser *under förutsättning att vissa villkor är uppfyllda*. Man *väljer ut* eller *selekterar* vilka satser som ska utföras.

Ex: I ovanstående exempel ska sats2 endast utföras om villkor1 är sant.

Strukturdiagram:



pseudokod:

```
Program
  sats1
  om villkor1
    sats2
  sats3
```

C-kod:

```
sats1;
if (villkor1)
   sats2;
sats3;
```

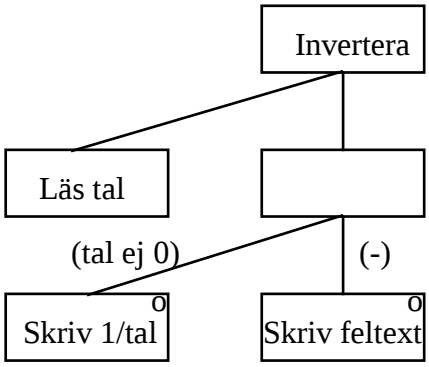
Villkor1 är ett logiskt uttryck som kan anta värdet falskt (0) eller sant (1). Då programmet kommer till if-satsen kontrollerar det värdet på villkor1. Är detta sant utförs sats2 annars inte.

OBS! I strukturdiagrammet ska *varje ruta ha en entydig struktur*. Den ska antingen vara sekvens, selektion eller iteration.

OBS! *Indragningen (indenteringen)* i pseudokoden och C-koden för att *markera att man styrt programflödet på ett speciellt sätt.*

Ex: Skriv ett program som läser in ett reellt tal och beräknar och skriver ut dess inverterade värde. Är det inlästa talet 0 ska feltexten 'Saknar inverterat värde!' skrivas ut.

Stukturdiagram:



pseudokod:

```
Invertera
  läs tal
  om tal ej 0
    skriv 1/tal
  annars
    skriv feltext
```

C-kod:

```
#include <stdio.h>

int main()
{
   double tal;

   printf("Ge tal: ");
   scanf("%lf", &tal);

   if (tal != 0)
      printf("Inverterat tal = %f\n", 1/tal);
   else
      printf("Saknar inverterat värde!\n");

   return 0;
}
```

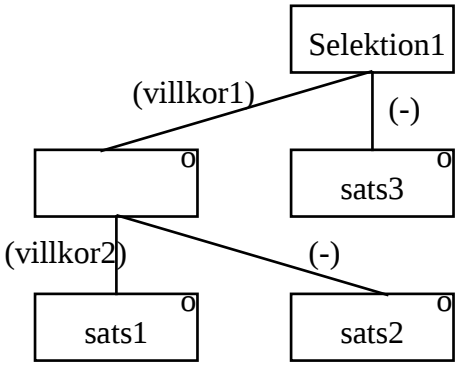
OBS! Annars markeras i strukturdiagrammet med ett streck (-).

OBS! Man kan om man vill skriva text i den tomma burken. I ovanstående exempel kan man exempelvis skriva bearbeta tal. Denna text kan sedan återkomma som rubrik i koden.

Selektioner kan nästlas d.v.s man kan ha selektioner inuti selektioner. Vid nästlade selektioner kan det uppstå problem med och veta till vilken del som annars-delen hör. Regeln är att annars-delen alltid hör till den *närmast föregående selektionen som saknar* annars-del.

Ex: En första nästlad selektion.

Strukturdiagram:

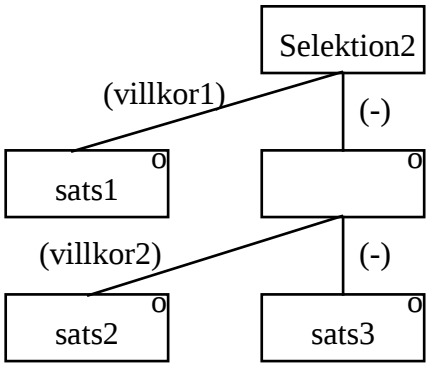


C-kod:

```
if (villkor1)
   if (villkor2)
      sats1;
   else
      sats2;
else
   sats3;
```

Ex: En andra nästlad selektion.

Strukturdiagram:



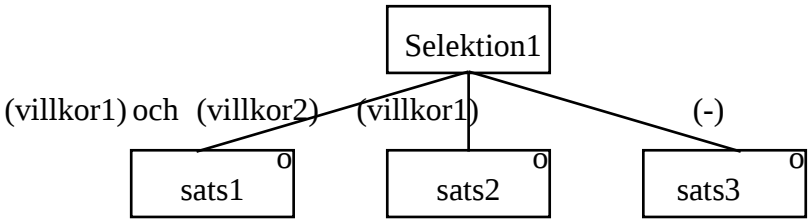
C-kod:

```
if (villkor1)
   sats1;
else
   if (villkor2)
      sats2;
   else
      sats3;
```

Nästlade selektioner har ofta en tendens att bli krångliga. Ser man att en selektion börjar bli väldigt djup i strukturdiagrammet bör man tänka om. Man ska då göra om selektions-villkoren så att en *flervalsselektion* fås med bredd istället för djup. I en flervalsselektion kan man välja mellan fler saker än två och endast en av sakerna utförs.

Ex: Skriv om den första selektionen ovan som en flervalsselektion.

Strukturdiagram:

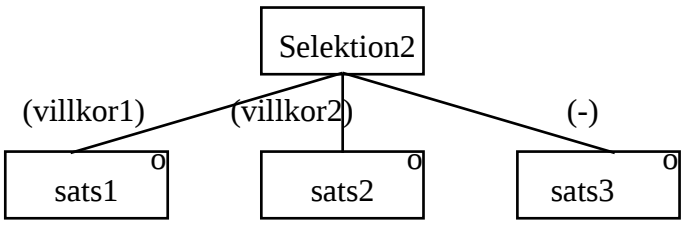


C-kod:

```
if (villkor1 && villkor2)
   sats1;
else if (villkor1)
   sats2;
else
   sats3;
```

Ex: Den andra selektionen skriven som en flervalsselektion.

Strukturdiagram:



C-kod:

```
if (villkor1)
   sats1;
else if (villkor2)
   sats2;
else
   sats3;
```

Ex: Skriv ett program som löser andragradsekvationen

$$x^2 + ax + b = 0$$

som har lösningarna

$$x = -a/2 \pm \sqrt{a^2/4 - b}$$

pseudokod:

```
Andragrad
  läs a, b
  c = a^2 / 4 - b
  om c > 0
    skriv x1
    skriv x2
  annars om c = = 0
    skriv x
  annars
    skriv 'Saknar reell lösning'
```

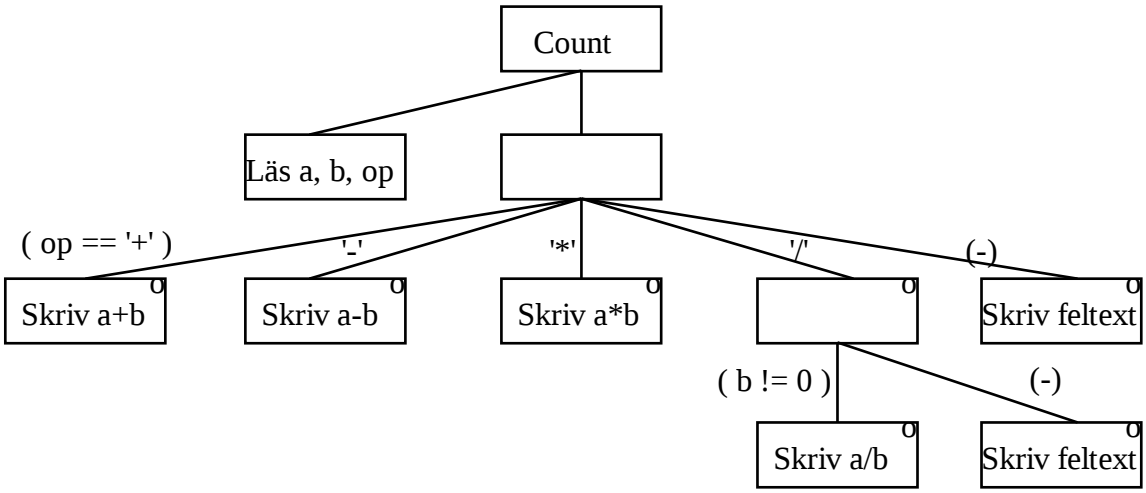
C-kod:

```
#include <stdio.h>
#include <math.h>

int main()
{
   double a, b, c;

   printf("Ge a och b: ");
   scanf("%lf%lf", &a, &b);

   c = pow(a, 2)/4 - b;

   if (c > 0)
   {                                          /* OBS! */
      printf("x1 = %f\n", -a/2 - sqrt(c));
      printf("x2 = %f\n", -a/2 + sqrt(c));
   }                                          /* OBS! */
   else if (c == 0)
      printf("x1 = x2 = %f\n", -a/2);
   else
      printf("Ekvationen saknar reell lösning!\n");

   return 0;
}
```

OBS! Ska flera satser utföras inuti en selektionsdel måste *blockparenteserna* { } användas.

Ex: Skriv ett program som läser in ett uttryck av typen 2.3 op 3.4 där op kan vara +, -, * eller /. Programmet ska sedan skriva ut resultatet av operationen. Matas en felaktig operator in ska ett felmeddelande skrivas ut. Ett felmeddelande ska också skrivas ut om man försöker dividera med 0.

Strukturdiagram:



C-kod:

```
#include <stdio.h>

int main()
{
   double a, b;
   char op;

   printf("Ge ett uttryck på formen a op b: ");
   scanf("%lf %c %lf", &a, &op, &b);

   if (op == '+')
      printf("%f\n", a + b);
   else if (op == '-')
      printf("%f\n", a - b);
   else if (op == '*')
      printf("%f\n", a * b);
   else if (op == '/')
      if (b != 0)
         printf("%f\n", a / b);
      else
         printf("Division med 0!\n");
   else
      printf("Fel operator!\n");

   return 0;
}
```

(Kom ihåg: Vid inläsning till en double med scanf ska man använda format-specificeraren ”`%lf`”, och vid utskrift med printf ska man använda ”`%f`”.)

Flervalsselektioner kan i vissa fall kodas med en *switch-sats* istället för en if-sats i C.

Ex: Skriv om programmet count ovan med en switch-sats istället för en if-sats.

```
#include <stdio.h>

int main()
{
   double a, b;
   char op;

   printf("Ge ett uttryck på formen a op b: ");
   scanf("%lf %c %lf", &a, &op, &b);

   switch (op)
   {
      case '+':
         printf("%f\n", a + b);
         break;                        /* OBS! */
      case '-':
         printf("%f\n", a - b);
         break;
      case '*':
         printf("%f\n", a * b);
         break;
      case '/':
         if ( b != 0)
            printf("%f\n", a / b);
         else
            printf("Division med 0!\n");
         break;
      default :
         printf("Fel operator!\n");
   }

    return 0;
}
```

Switch-satsen fungerar så att satsens styruttryck, som ovan är variabeln op och som måste vara uppräknelig (heltal), beräknas. Därefter hoppar programmet in i den första case-del som överenstämmer med detta värde och *alla efterföljande satser utförs*. Om det ej finns något överenstämmande värde utförs satserna i default-delen.

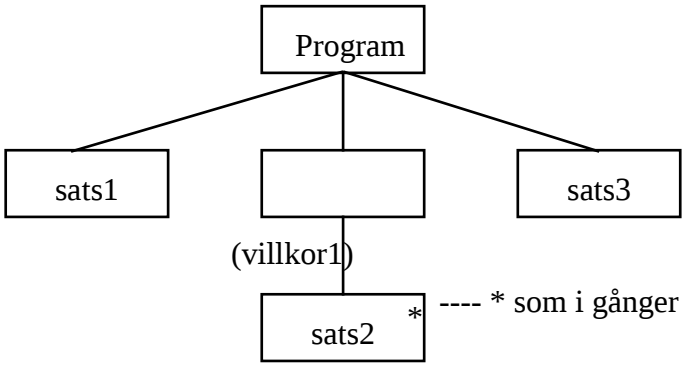
OBS! *Man måste hoppa ur switch-satsen med break*. Har man ej med break kommer alla efterföljande delar att också utföras. Har man exempelvis läst in ett *-tecken kommer programmet att utföra multiplikation, division och även skriva ut ett felmeddelande om felaktig operator om den aktuella case-delen skulle sakna break.

3.2 Iteration

Med styrmekanismen *iteration*, även kallad *loop* eller *slinga*, kan man välja att upprepa satser *så länge något villkor är uppfyllt*.

Ex: I ovanstående första exempel ska sats2 upprepas så länge villkor1 är sant.

Strukturdiagram:



pseudokod:

```
Program
  sats1
  så länge villkor1
    sats2
  sats3
```

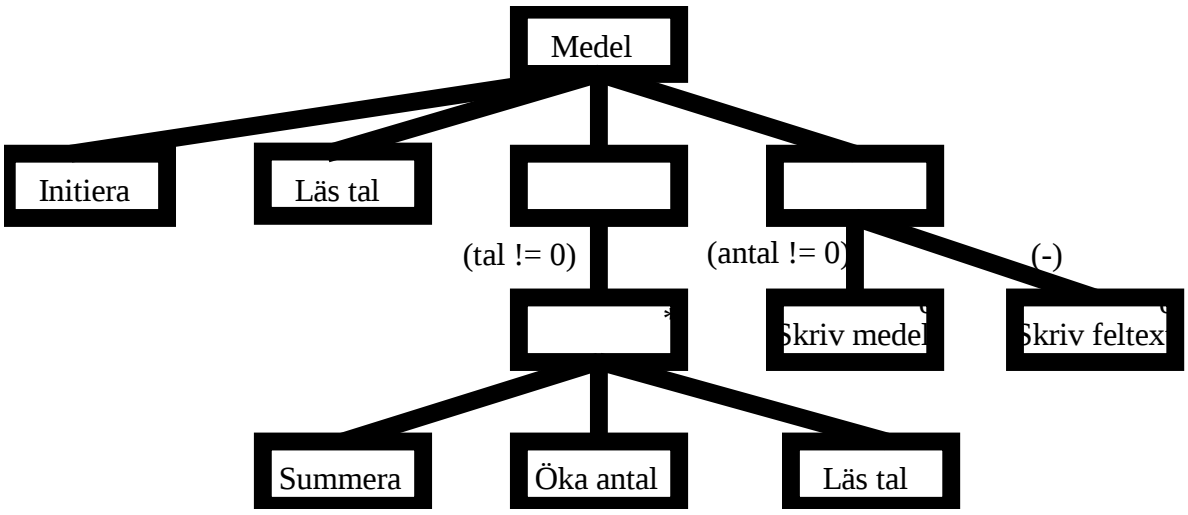
C-kod:

```
sats1;
while (villkor1)
   sats2;
sats3;
```

Villkor1 är ett logiskt uttryck som kan anta värdet falskt (0) eller sant (1). Då programmet kommer till while-satsen kontrollerar det värdet på villkor1. Är detta sant utförs sats2 och ny kontroll av villkor1 görs o.s.v. Då villkoret blir falskt fortsätter programmet med sats3.

Ex: Skriv ett program som läser in tal och summerar talen så länge de inlästa talet är skilt ifrån 0. Efter inläsning ska medelvärdet skrivas ut om något tal lästs in annars feltext.

Strukturdiagram:



C-kod:

```
#include <stdio.h>

int main()
{
   double tal, summa = 0.0;
   int antal = 0;

   printf("Ge tal (avsluta med 0): ");
   scanf("%lf", &tal);

   while (tal != 0)
   {                                          /* OBS! */
      summa += tal;
      antal++;
      printf("Ge tal (avsluta med 0): ");
      scanf("%lf", &tal);
   }                                          /* OBS! */

   if (antal != 0)
      printf("Medel = %f\n", summa / antal);
   else
      printf("Finns inget medel att beräkna!\n");

   return 0;
}
```

OBS! Om flera satser ska upprepas måste *blockparenteserna* { } användas.

En *verklig iteration ska kunna upprepas 0, 1 eller flera gånger*. Ibland kanske man vill att en sats alltid ska utföras minst en gång och sedan eventuellt upprepas så länge ett villkor är sant. I C finns en speciell konstruktion för detta som heter *do ... while*.

Ex: I vårt arbetsexempel vet man att sats2 ska köras minst en gång och sedan upprepas så länge villkor1 är sant.

pseudokod:

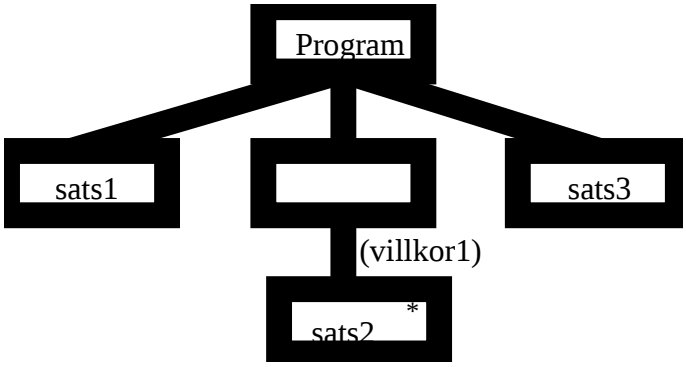
```
Program
  sats1
  gör
    sats2
  så länge (villkor1)
  sats3
```

C-kod med do .. while:

```
sats1;
do
   sats2;
while (villkor1);
sats3;
```

Strukturdiagram:

I strukturdiagrammet markerar man med texten do .. while (villkor1) istället för med bara while.



Konstruktionen `do` ... `while` kan till exempel vara användbar när programmet ska läsa in data, och man vill upprepa inläsningen tills användaen matar in ett tillåtet värde.

Ex: Ekvationen $x^3 - 25x^2 + 18x - 450 = 0$ har en lösning som är ett litet positivt heltal. Skriv ett program som hittar denna lösning genom att pröva med talen 1, 2, 3 o.s.v.

C-kod med while:

```
#include <stdio.h>
#include <math.h>

int main()
{
   int y, x = 0;

   x++;
   y = pow(x, 3) - 25* pow(x, 2) + 18*x - 450;

   while (y != 0)
   {
      x++;
      y = pow(x, 3) - 25* pow(x, 2) + 18*x - 450;
   }

   printf("x = %d\n", x);

   return 0;
}
```

C-kod med do .. while:

```
#include <stdio.h>
#include <math.h>

int main()
{
   int y, x = 0;

   do
   {
      x++;
      y = pow(x, 3) - 25* pow(x, 2) + 18*x - 450;
   }
   while (y != 0);

   printf("x = %d\n", x);

   return 0;
}
```

Ex: Från en *meny ska man upprepat kunna välja olika alternativ*. I menyn ska finnas ett alternativ som man kan avsluta upprepningen med. Skriv ett program som upprepat visar menyn:

K Kvadrat
R Rot
S Sluta

Välj >>

mitt på en tom skärm. Programmet ska kunna läsa reella tal, skriva ut kvadraten eller kvadratroten mitt på skärmen.

```
#include <stdio.h>
#include <math.h>

/* Makro för att rensa bufferten */
#define SKIPLINE while (getchar() != '\n')

int main()
{
   double tal;
   char svar;

   do
   {
      /* menytext */
      printf("K   Kvadrat\n");
      printf("R   Roten\n");
      printf("S   Sluta\n");
      printf("\n");
      printf("Välj >> ");

      /* läs svar och rensa bufferten */
      svar = getchar();
      SKIPLINE;
```

```
        switch (svar)
        {
           case 'k':
           case 'K':
              /* läs tal och rensa */
              printf("Ge ett tal: ");
              scanf("%lf", &tal);
              SKIPLINE;

              /* skriv kvadraten */
              printf("Kvadraten: %f\n", pow(tal, 2));
              printf("Tryck RETURN!!\n");
              SKIPLINE;
              break;

           case 'r':
           case 'R':
              /* läs tal och rensa */
              printf("Ge ett tal: ");
              scanf("%lf", &tal);
              SKIPLINE;

              /* skriv roten */
              if ( tal >= 0 )
                 printf("Kvadratroten: %f\n", sqrt(tal));
              else
                 printf("Kvadratroten ej definierad!\n");
              printf("Tryck RETURN!!\n");
              SKIPLINE;
              break;

           case 's':
           case 'S':
              /* gör ingenting */
              break;
           default:
              /* skriv feltext */
              printf("Fel val!!\n");
              printf("Tryck RETURN!!\n");
              SKIPLINE;
        }
     }
     while (svar != 's' && svar != 'S');

     return 0;
  }
```

(När vi senare ser hur man kan dela upp sitt program i funktioner, kommer vi att se bättre alternativ till det där SKIPLINE-makrot.)

Ibland kanske man vill att en sats ska *upprepas ett visst antal gånger*. Istället för att använda en räknare som initieras och räknas upp så länge den är mindre än antalet kan man i C använda en konstruktion med *for*.

Ex: I vårt arbetsexempel vill man att sats2 ska upprepas 20 gånger.

pseudokod:

```
Program
  sats1
  nr = 1
  så länge (nr <= 20)
    sats2
    nr++
  sats3
```

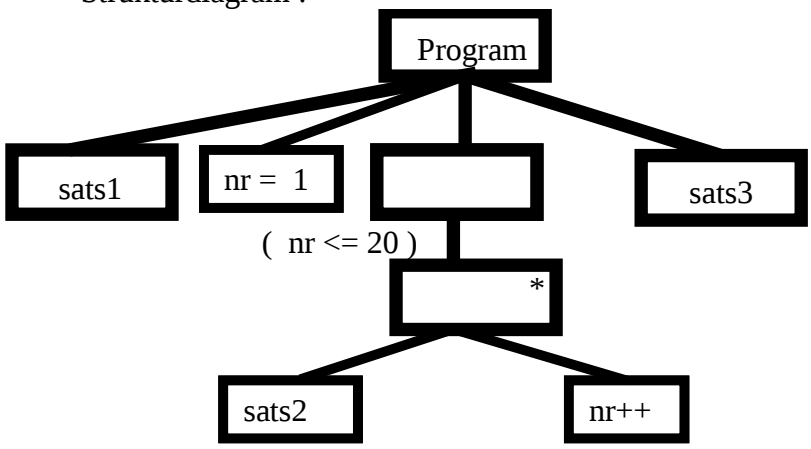
C-kod med while-konstruktion:

```
int nr;

sats1;
nr = 1;
while (nr <= 20)
{
   sats2;
   nr++;
}
sats3;
```

C-kod med for-konstruktion:

```
int nr;

sats1;
for(nr = 1; nr <= 20; nr++)
   sats2;
sats3;
```

Strukturdiagram :



Ex: Skriv ett program som frågar efter ett positivt heltal och skriver ut multiplikationstabellen för talet. Matar man exempelvis in talet 5 ska utskriften bli:

```
0 * 5 = 0
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
```

C-kod med while-sats:

```
#include <stdio.h>

int main()
{
   int tal, mult;

   printf("Ge talet: ");
   scanf("%d", &tal);

   mult = 0;
   while (mult <= tal)
   {
      printf("%d * %d = %d\n", mult, tal, mult*tal);
      mult++;
   }

   return 0;
}
```

C-kod med for-sats:

```
#include <stdio.h>

int main()
{
   int tal, mult;

   printf("Ge talet: ");
   scanf("%d", &tal);

   for (mult = 0; mult <= tal; mult++)
      printf("%d * %d = %d\n", mult, tal, mult*tal);

   return 0;
}
```

Man kan naturligvis ha nästlade iterationer precis som man kan ha nästlade selektioner. Man pratar om yttre och inre iteration. Den inre iterationen kommer att upprepas *fullständigt* varje gång som den yttre iterationen upprepas.

Ex: Skriv ett program som tillverkar ett mönster på skärmen enligt:

```
XXXXXXXXXXXXXXXXX
XXXXXXXX XXXXXXXX
XXXXXXX   XXXXXXX
XXXXXX     XXXXXX
XXXXX       XXXXX
XXXX         XXXX
XXX           XXX
XX             XX
X               X
```

```
#include <stdio.h>
int main()
{
   int rad, kol;

   /* för alla rader */
   for (rad = 1; rad <= 9; rad++)
   {
      /* för alla kolumner */
      for (kol = 1; kol <= 17; kol++)
      {
         if (kol > (10 - rad) && kol < (rad + 8))
            printf(" ");
         else
            printf("X");
       }
      printf("\n");
   }

   return 0;
}
```

OBS! I en nästlad loop körs alltid den innersta loopen färdigt innan man går till den yttre loopen och kör nästa varv. I exemplet ovan innebär det att man för varje rad kör färdigt alla kolumner innan man går till nästa rad.

OBS! I exemplen ovan har vi låtit loop-variablerna till exempel stegas från 1 till 17, om vi vill köra loopen 17 gånger. I C är det vanligare att man i så fall låter loop-variablerna stegas från 0 till 16. Men mer om det senare.

3.3 Övriga styrmekanismer

Man kan även styra programflödet med hjälp av hopp. I C finns möjligheter att hoppa till en angiven plats i programmet med goto.

Ex:

```
.....
if (villkor1)
   goto slut;
.....
slut: printf("Ett fel har uppstått. Vi måste avbryta!");
```

Här finns slut som sista sats i programmet så att programmet avsutas om villkor1 är uppfyllt. Man kunde ha uppnått samma sak genom att använda funktionen exit i process.h enligt:

```
if (villkor1)
{
   printf("Ett fel har uppstått. Vi måste avbryta!");
   exit(1);
}
```

Man kan hoppa ur en *iteration* eller en selektion i form av en *switch-sats* med hjälp av break.

Ex:

```
.....
while (villkor1)
{
   .....
   if (villkor2)
      break;
}
.....
```

Här hoppar programmet ur while-loopen om villkor2 är sant.

Man kan hoppa över en upprepning med continue.

Ex:

```
.....
for (nr = 0; nr <= 10; nr++)
{
   if (nr == 5)
      continue;
   sats1;
}
```

Här utföres sats1 ej när nr är 5 utan man fortsätter direkt med nr = 6.

OBS! **Hopp försvårar programunderhåll. Försök att undvika hopp i största möjliga utsträckning. Använd istället strukturerna sekvens, selektion och iteration.**