Språket för inbyggda system 2

C

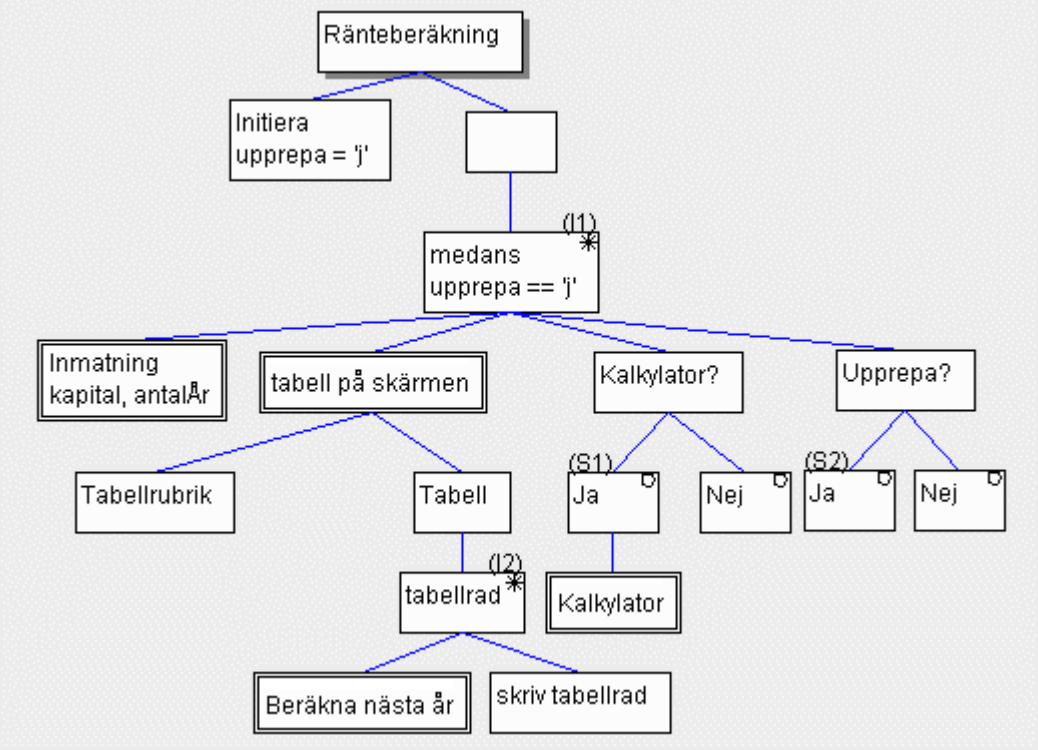
Ett programmeringsprojekt,

flera personer utvecklar ett program

LADYSOFT - programming

Vi ska utveckla ett ränteberäkningsprogram !

Programmets struktur *modulariserad programmering*



Anders Sjögren as@kth.se

Preprocessorn
något om dess användning i projektet

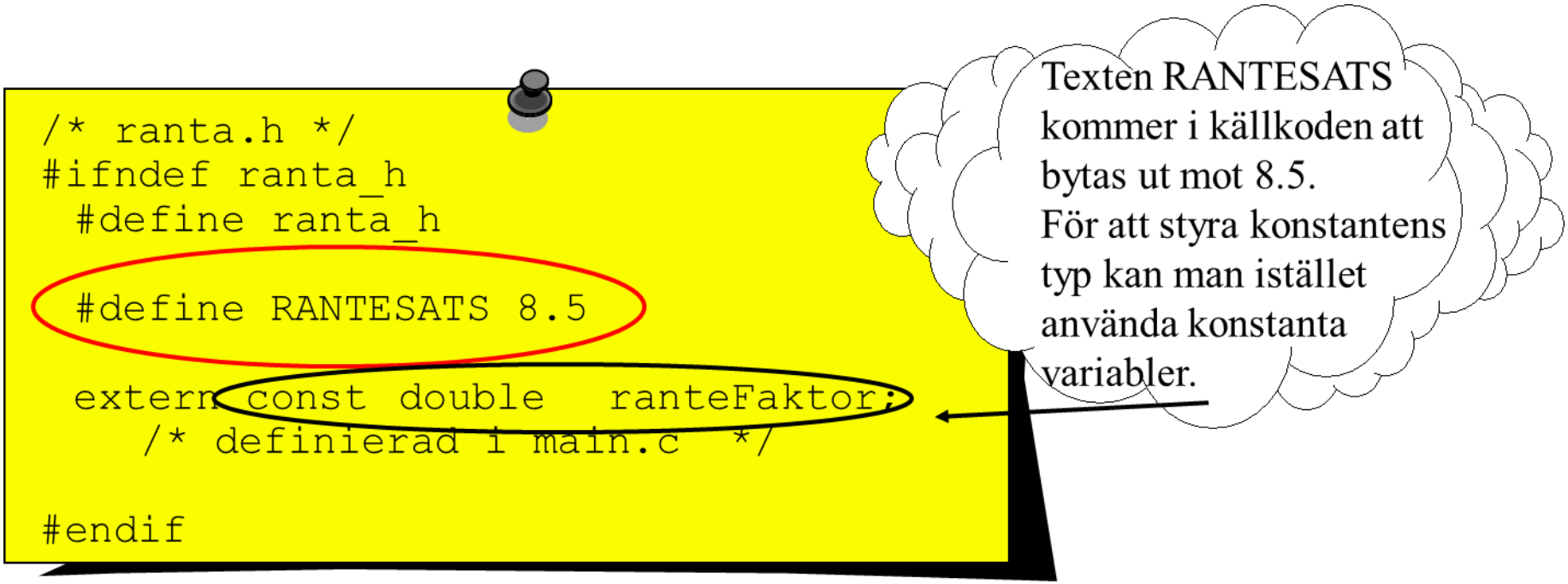
Preprocessorn är ett textmanipuleringsinstrument som används/utförs före kompilering

- alla preprocessordirektiv föregås av # - tecknet
- de vanligaste preprocessovarianterna är
  - makro utan argument - ”konstantdefinition”
  - makro med argument
  - filinkludering
  - villkorligt medtagande/uteslutande av text

Anders Sjögren as@kth.se

Konstantdefinition

- de vanligaste preprocessovarianterna är
  - makro utan argument - ”konstantdefinition”



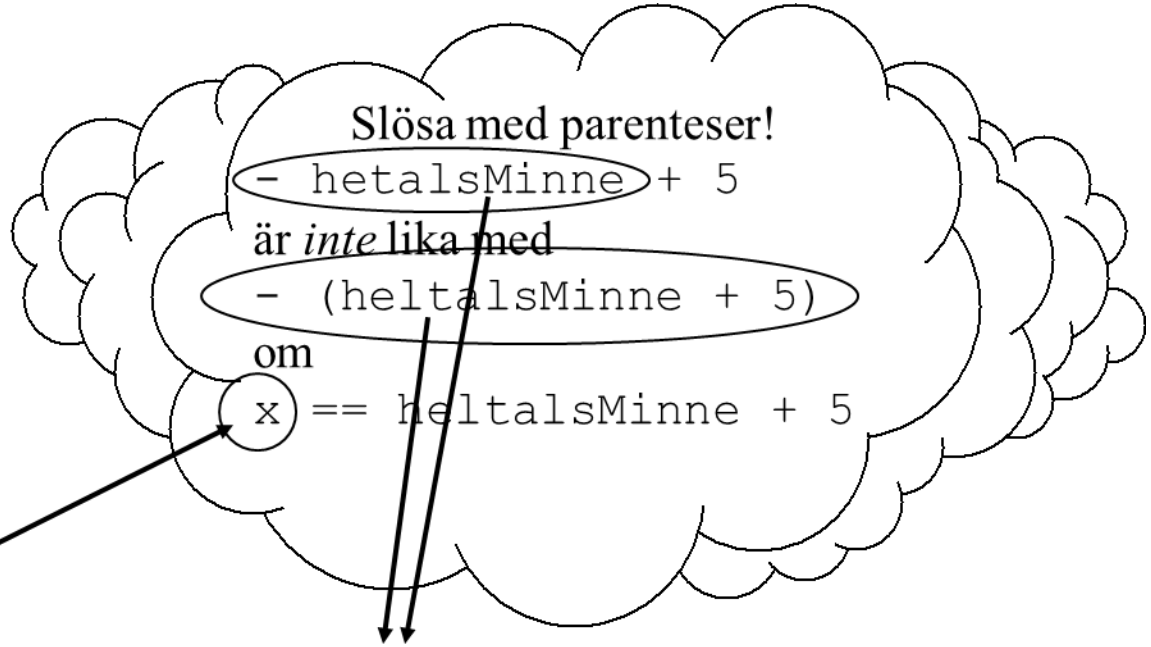
Makro med argument

- de vanligaste preprocessorvarianterna är
  - makro med argument

Makro med argument liknar funktioner. De används för att förenkla uttryck som anses ”för små” för att göra funktion av. Man kortar exekverings-tiden pga att inget funktions-anrop görs ty uttrycket expanderas i källkods-texten.

```
/* tabell.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(tabell_h)
   #define tabell_h

   #define ABS( x )( (x) > 0 ? (x) : -(x) )
   void TabellPaSkarmen( double kapital, int antalAr );

#endif
```

Parenteser!




```
/* tabell.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(tabell_h)
   #define tabell_h

   #define ABS( x )( (x) > 0 ? (x) : -(x) )
   void TabellPaSkarmen( double kapital, int antalAr );

#endif
```

Anders Sjögren as@kth.se

Fil-inkludering

- de vanligaste preprocessorvarianterna är
  - filinkludering

```
/* kalkylat.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(kalkylat_h)
  #define kalkylat_h

  void Kalkylator( void );

#endif
```

Angivna textfilers innehåll kommer att expanderas här. ” ” betyder att filen hämtas från en egen katalog medans < > betyder att filen hämtas från en i utvecklingsmiljön fördefinierad katalog.

villkorligt medtagande/uteslutande

- de vanligaste preprocessovarianterna är
  - villkorligt medtagande/uteslutande av text

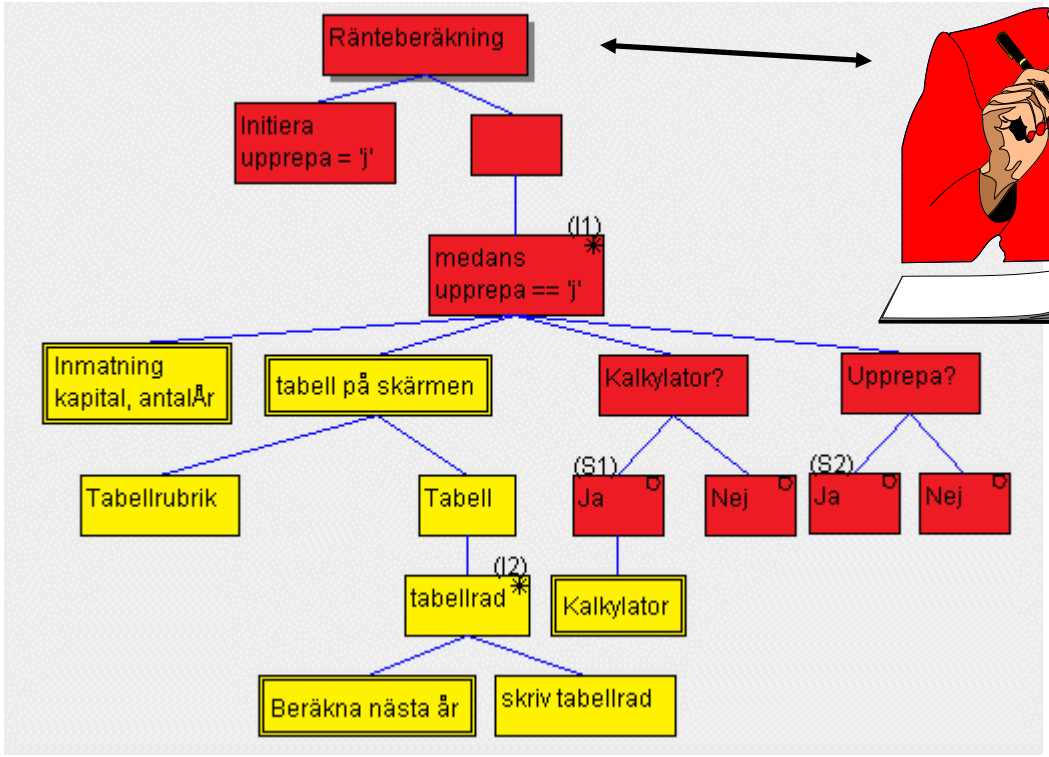
```
/* kalkylat.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(kalkylat_h)
  #define kalkylat_h

  void Kalkylator( void );

#endif
```

För att förhindra att viss text kommer med på flera ställen kan denna konstruktion användas.

kalkylat_h används som en flagga att testa på.

Programmets struktur

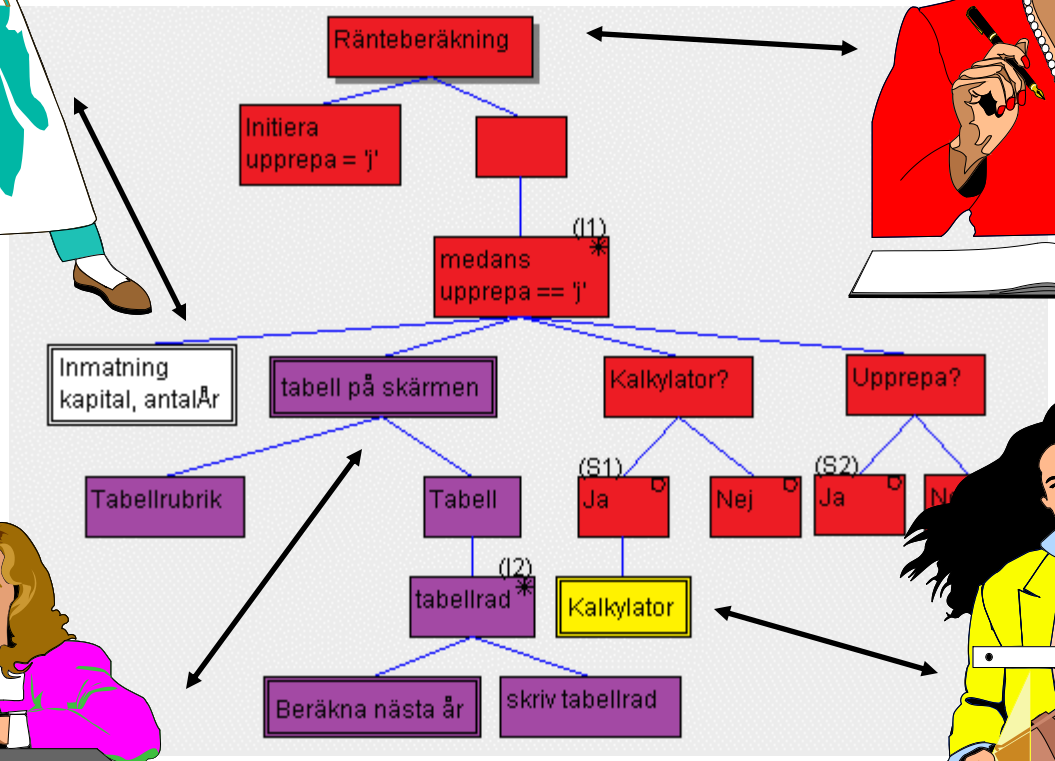
arbetsfördelning

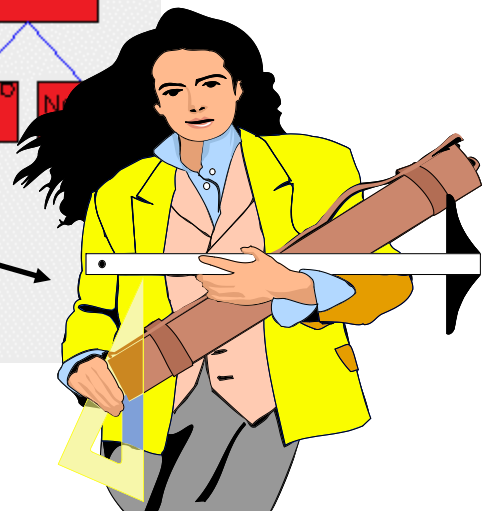


Anders Sjögren as@kth.se

Programmets struktur

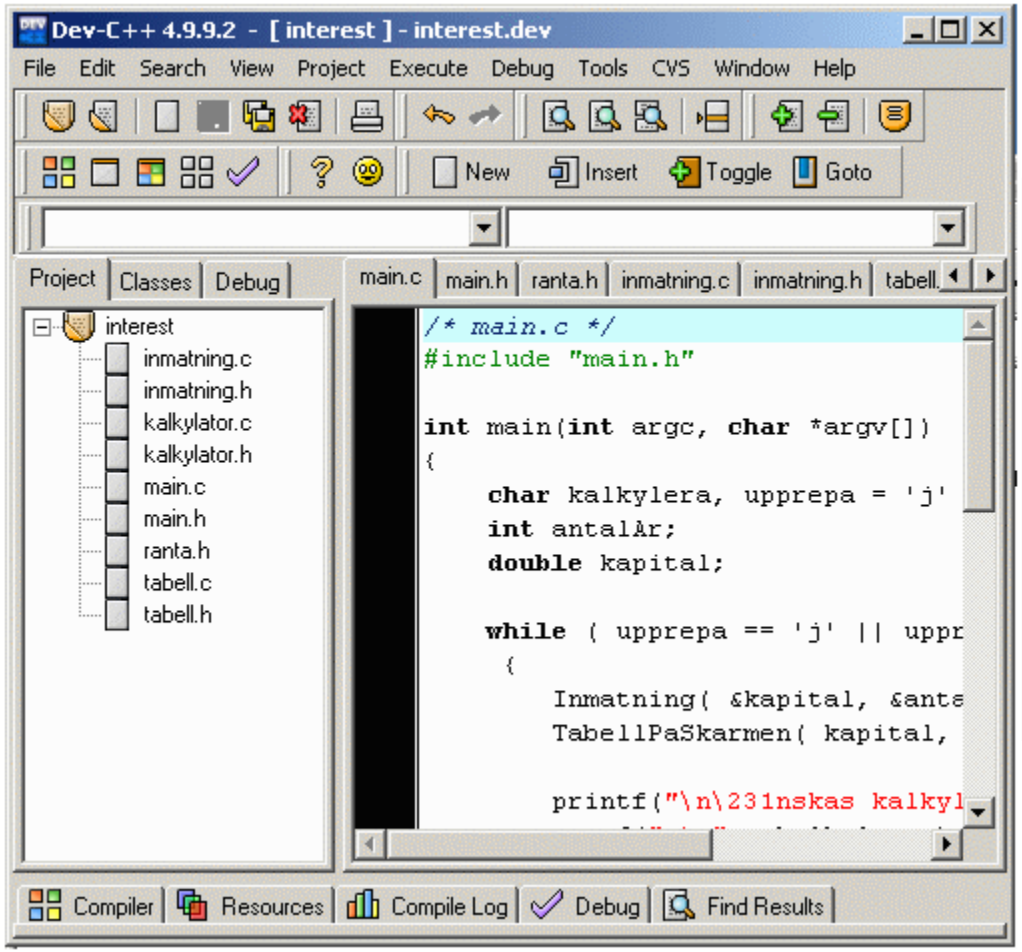
arbetsfördelning





Anders Sjögren as@kth.se

Filer i projektet

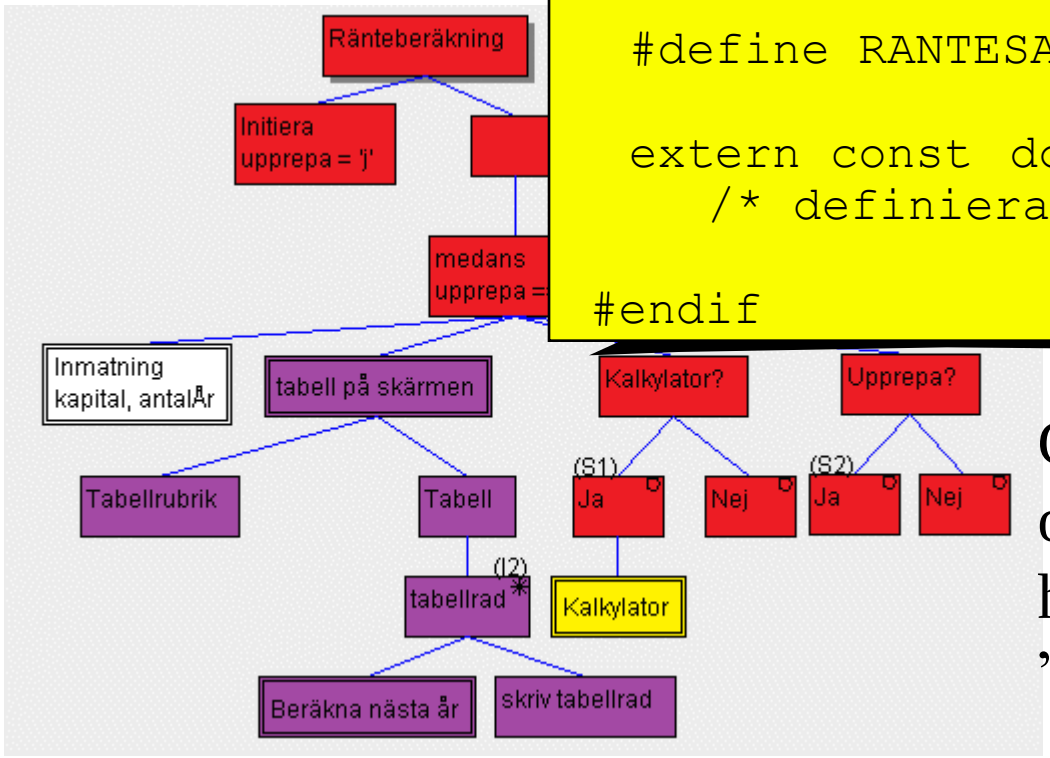


Anders Sjögren as@kth.se

Programmets struktur

gemensam *headerfil*

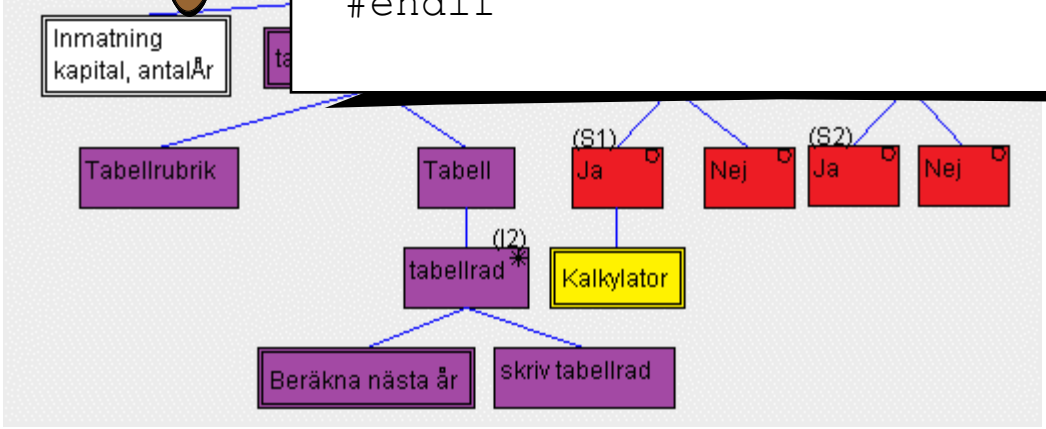
```
/* ranta.h */
#ifndef ranta_h
  #define ranta_h

  #define RANTESATS 8.5

  extern const double   ranteFaktor ;
     /* definierad i main.c  */

#endif
```



Gemensamma deklarationer och definitioner samlas i en header-fil som alla ”inkluderar”.

Arbetsfördelning och filer

```
/* inmatnin.h */
#include "ranta.h"
#include <stdio.h>
#include <conio.h> /* ej ANSI, clrscr() används */

#if !defined(inmatnin_h)
  #define inmatnin_h

  void Inmatning(double* kapitalPek,int*antalArPek);

#endif
```



Varje projektdeltagare skriver sina egna filer som kan kompileras separat för att kontrollera syntaxen i den egna koden.

Funktionsdeklarationer och makron i h-filen(er) och funktionsdefinitioner i c-filen(er).

Arbetsfördelning och filer

```
/* inmatnin.c */
#include "inmatnin.h"

void Inmatning( double* kapitalPek, int* antalArPek ){
  clrscr();   /* ej ANSI-C */
  printf("\nBeräknar kapitaltillväxt vid %0.1f ränta",RANTESATS)
  printf("\n=========================================");
  printf("\nPositivt kapital räknar framåt i tiden.");
  printf("\nNegativt kapital räknar bakåt i tiden.");
  printf("\n\nInsatt kapital och antal år ? ");
  printf("(-->(+/-)1000 10)-->");
  scanf("%lf%d", kapitalPek, antalArPek);

  return;
}
```

Inmatning kapital, antalÅr

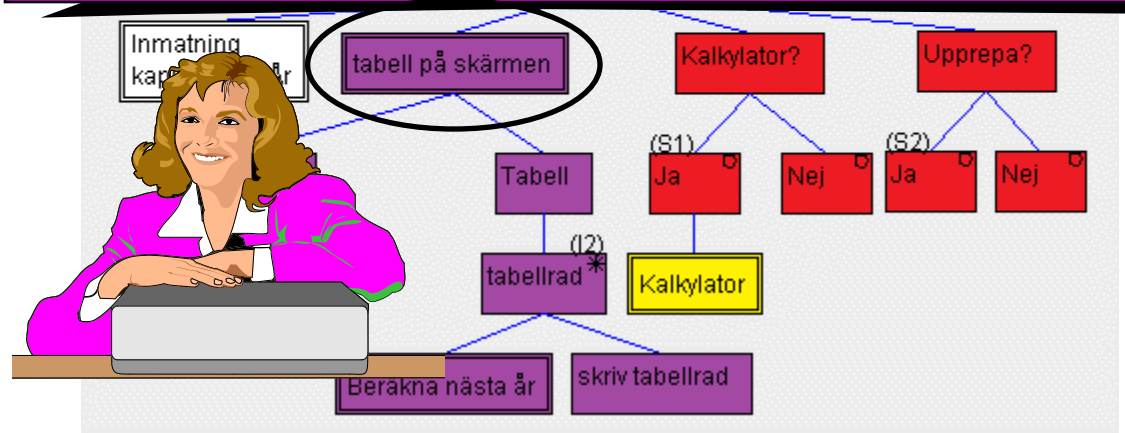
Tabellrubrik

Beräkna nästa år

Anders Sjögren as@kth.se

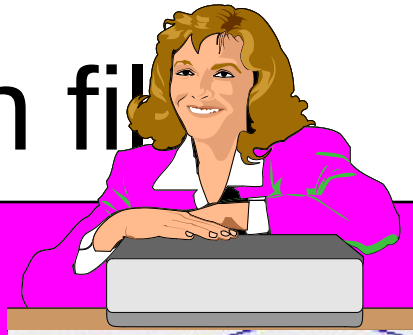
Arbetsfördelning och filer

```
/* tabell.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(tabell_h)
  #define tabell_h

    #define ABS( x )( (x) > 0 ? (x) : -(x) )
  void TabellPaSkarmen( double kapital, int antalAr );

#endif
```



Anders Sjögren as@kth.se

Arbetsfördelning och fil



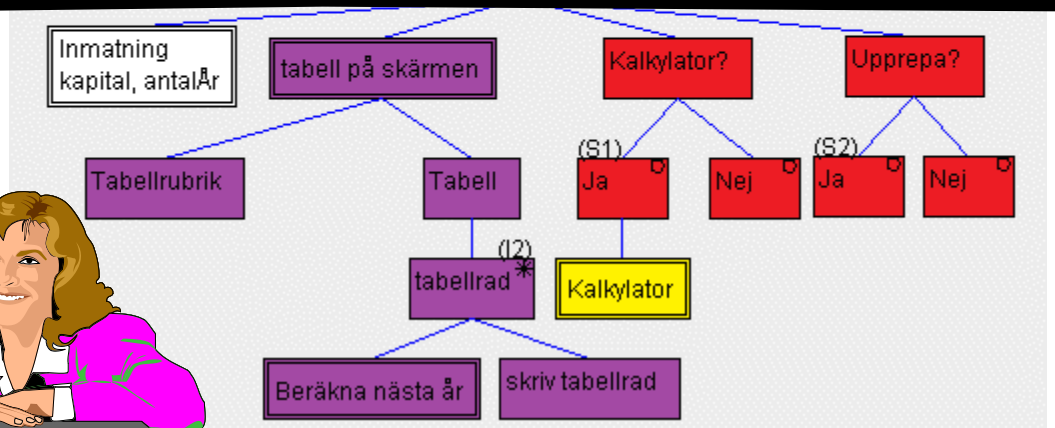

```
#include "tabell.h"

static double NastaAr( double kapital );

void TabellPaSkarmen( double kapital, int antalAr   )
{
  int ar ;

  printf("\n År      Saldo\n ==      =====\n");
  for ( ar = 1; ar <= antalAr; ar++ ) {
    kapital = NastaAr( kapital );

    /* enheter i tabellen */
    if ( -10 < kapital && kapital < 10 )
      printf("%3d%11.2f kr\n", ar, ABS( kapital ));
    else if ( -100 < kapital && kapital < 100 )
      printf("%3d%11.2f da(deka)kr\n", ar, (ABS( kapital ))/10);
    else if ( -1000 < kapital && kapital < 1000 )
      printf("%3d%11.2f h(hekto)kr\n", ar, (ABS( kapital ))/100);
    else
      printf("%3d%11.2f kkr\n", ar, (ABS( kapital ))/1000);
  }
  return;
}
```

Forts ...

Arbetsfördelning och filer

Forts ...

```
static double NastaAr( double x ) {

  if ( x > 0 )
    x = x * ( 1 + ranteFaktor );          /* denna */
  else  x = x * 1/( 1 + RANTESATS/100 ); /* eller denna */

  return x ;
}
```



Anders Sjögren as@kth.se

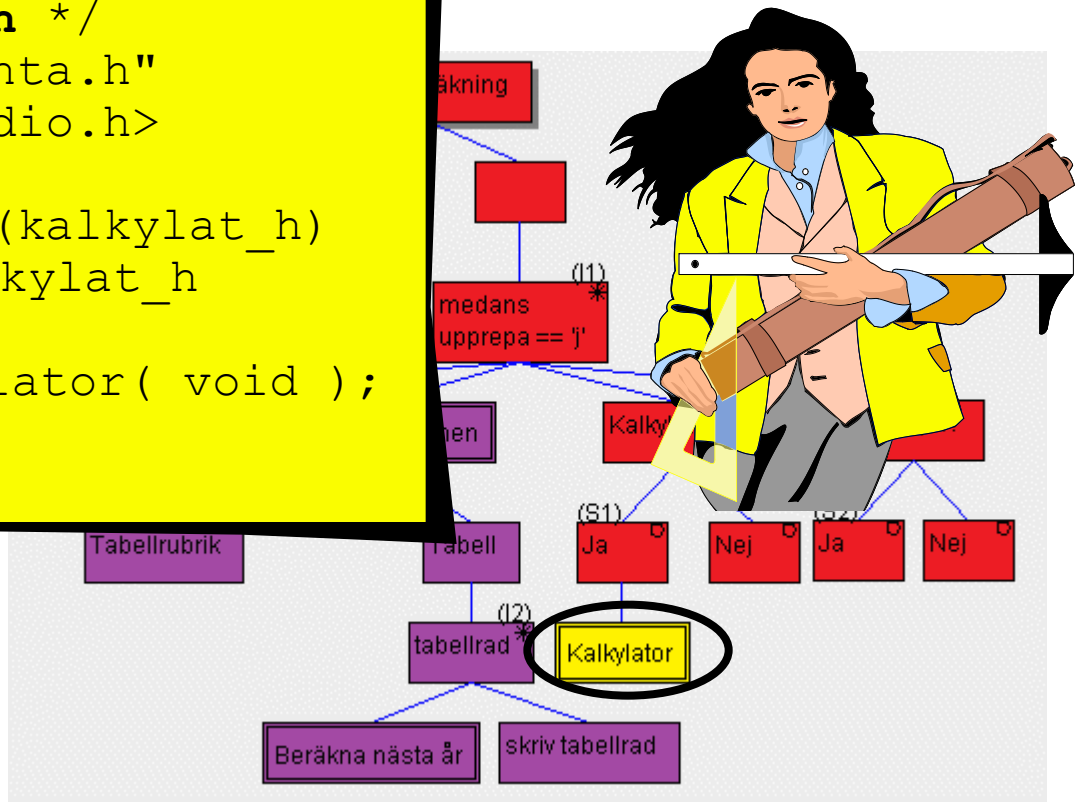
Arbetsfördelning och filer

Forts …

```
static double NastaAr( double x ) {

  if ( x > 0 )
    x = x * ( 1 + ranteFaktor );          /* denna */
  else  x = x * 1/( 1 + RANTESATS/100 ); /* eller denna */

  return x ;
}
```

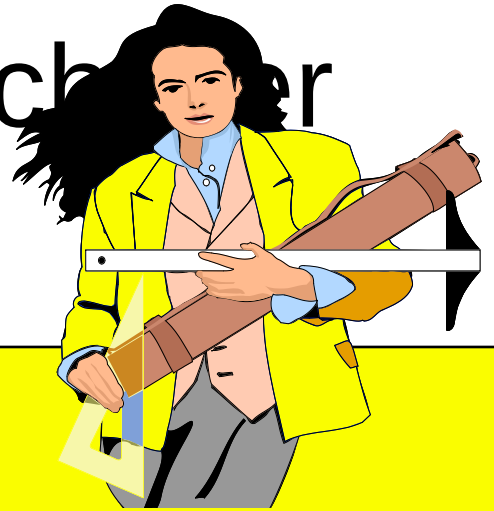
Arbetsfördelning och filer

```
/* kalkylat.h */
#include "ranta.h"
#include <stdio.h>

#if !defined(kalkylat_h)
  #define kalkylat_h

  void  Kalkylator( void );

#endif
```



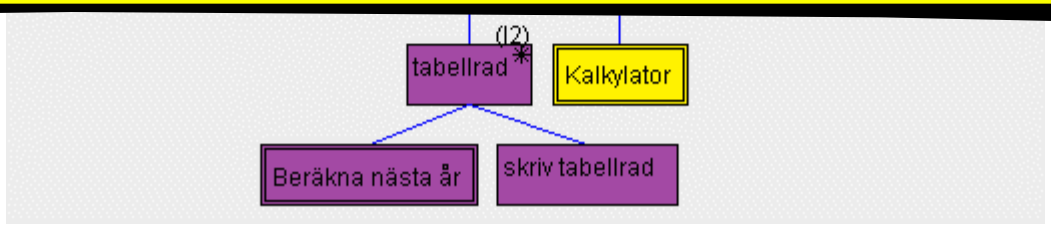
Anders Sjögren as@kth.se

Arbetsfördelning och

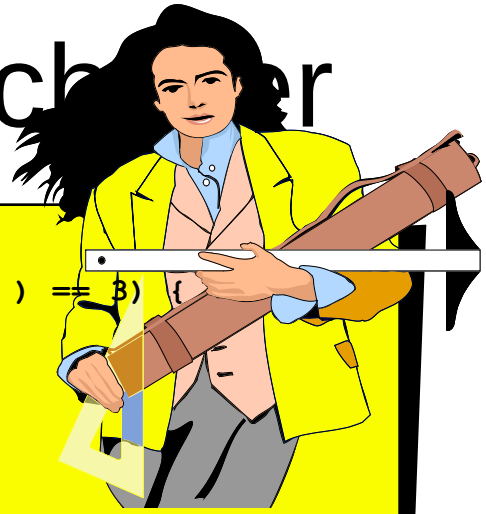


```
/* kalkylat.c */
#include "kalkylat.h"

void Kalkylator( void )    /* Enkel kalkylator */
{
    float x, y;
  char c;

  printf( "\nKalkylator som klarar de fyra räknesätten t ex ");
  printf("3+2\n");
  printf( "A, avslutar\n");
```

forts



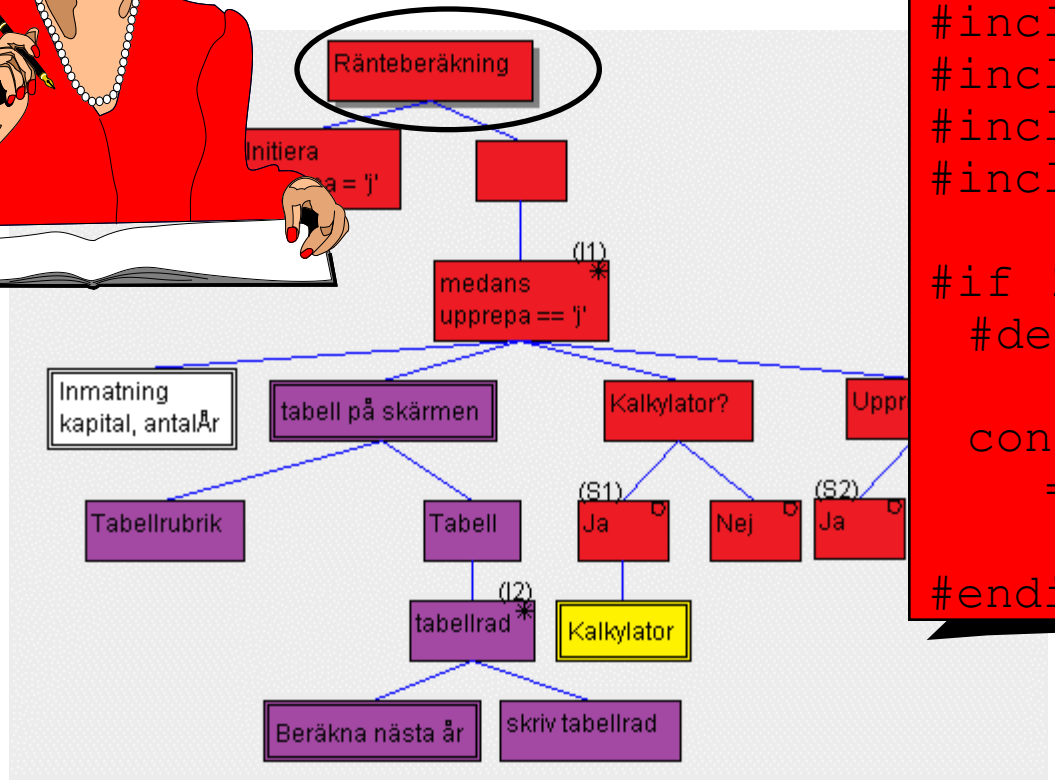
Arbetsfördelning och ...er



forts

```
while (printf("-->"), scanf("%f%c%f", &x, &c, &y ) == 3) {
  switch(c) {
      case '+':
        printf("%f\n", x + y);
      break;
      case '-':
        printf("%f\n", x - y);
        break;
    case '*':
        printf("%f\n", x * y);
     break;
    case '/':
        if (y != 0)
          printf("%f\n", x / y);
        else
           printf("Division med noll\n");
        break;
      default:
        printf("Felaktig operator\n");
        break;
    }
  }
  scanf("%*s");
  return;
}
```

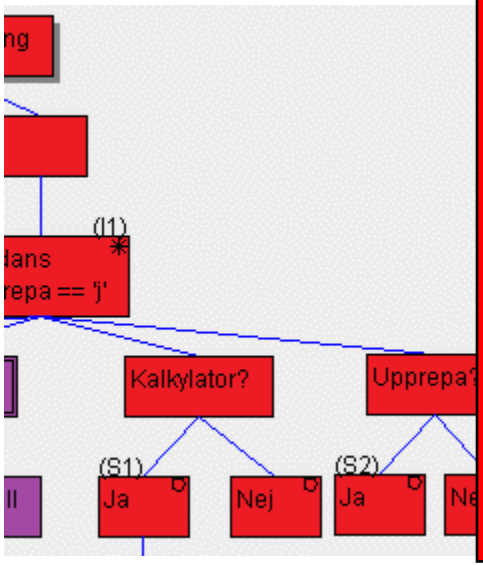
Arbetsfördelning och filer




```
/* main.h */
#include "ranta.h"
#include "inmatnin.h"
#include "tabell.h"
#include "kalkylat.h"
#include"ranta.h"
#include <stdio.h>

#if !defined(main_h)
  #define main_h

  const double ranteFaktor
     = RANTESATS / 100;

#endif
```

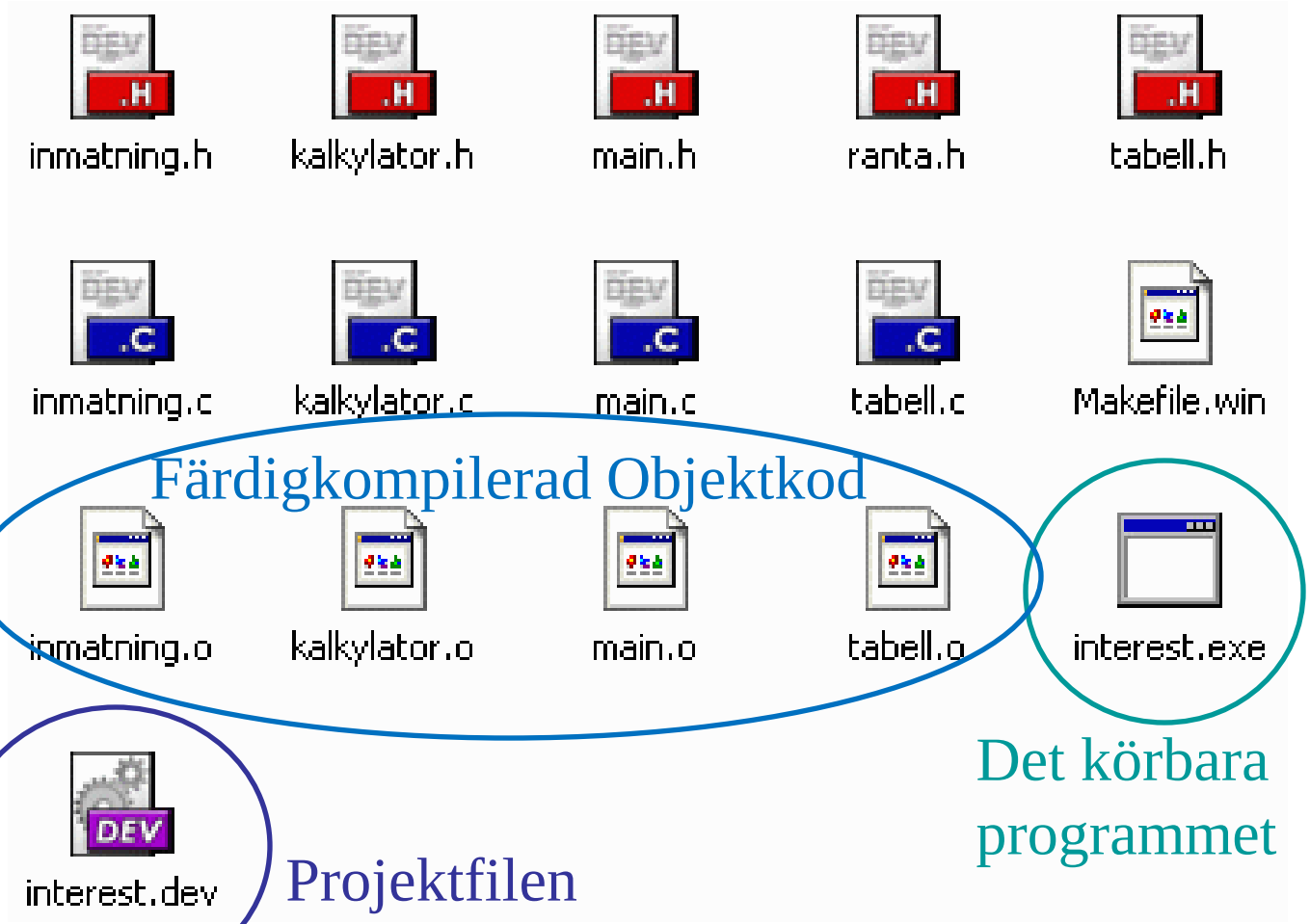
Anders Sjögren as@kth.se

Arbetsfördelning och filer

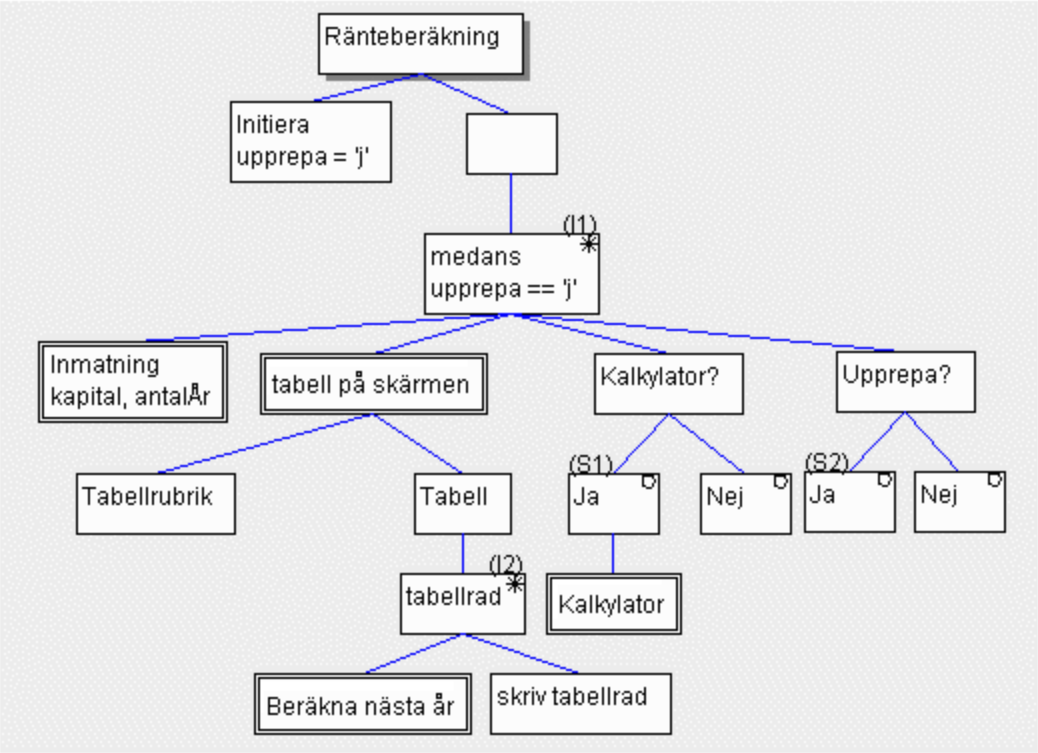



```
/* main.c */
#include "main.h"

int main( void ) {

  char   kalkylera, upprepa = 'j' ;
  int  antalAr;
  double   kapital;

  while ( upprepa == 'j' || upprepa == 'J' ) {
    Inmatning( &kapital, &antalAr );
    TabellPaSkarmen( kapital, antalAr );

    printf("\nÖnskas kalkylator? ( j/n ) --> ");
    scanf(" %c", &kalkylera );
    if ( kalkylera == 'j' || kalkylera == 'J' )
      Kalkylator();

    printf("\nUpprepa programmet? (j/n) --> ");
    scanf(" %c", &upprepa);
  }
  printf("\nSLUT");
  return 0;
}
```

Filer i projektet



Bygg projektet själv



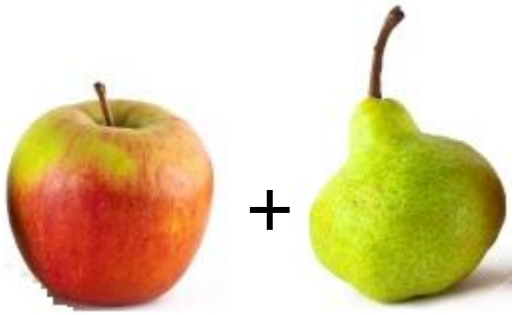
Alla källkodstexter finns på ID120V kurswebben …

`http://www.ict.kth.se/courses/ID120V/files/funktioner/project.htm`

William Sandqvist william@kth.se

Typecast av variabler

Man kan ju ***inte*** lägga ihop äpplen med päron – men C gör i så fall det bästa (?) möjliga av situationen.



C gör automatiska typomvandlingar :

double int float int

```
d = i + f/100
```

William Sandqvist william@kth.se

Typomvandling implicit (automatisk)

```
d = i + f/100
```

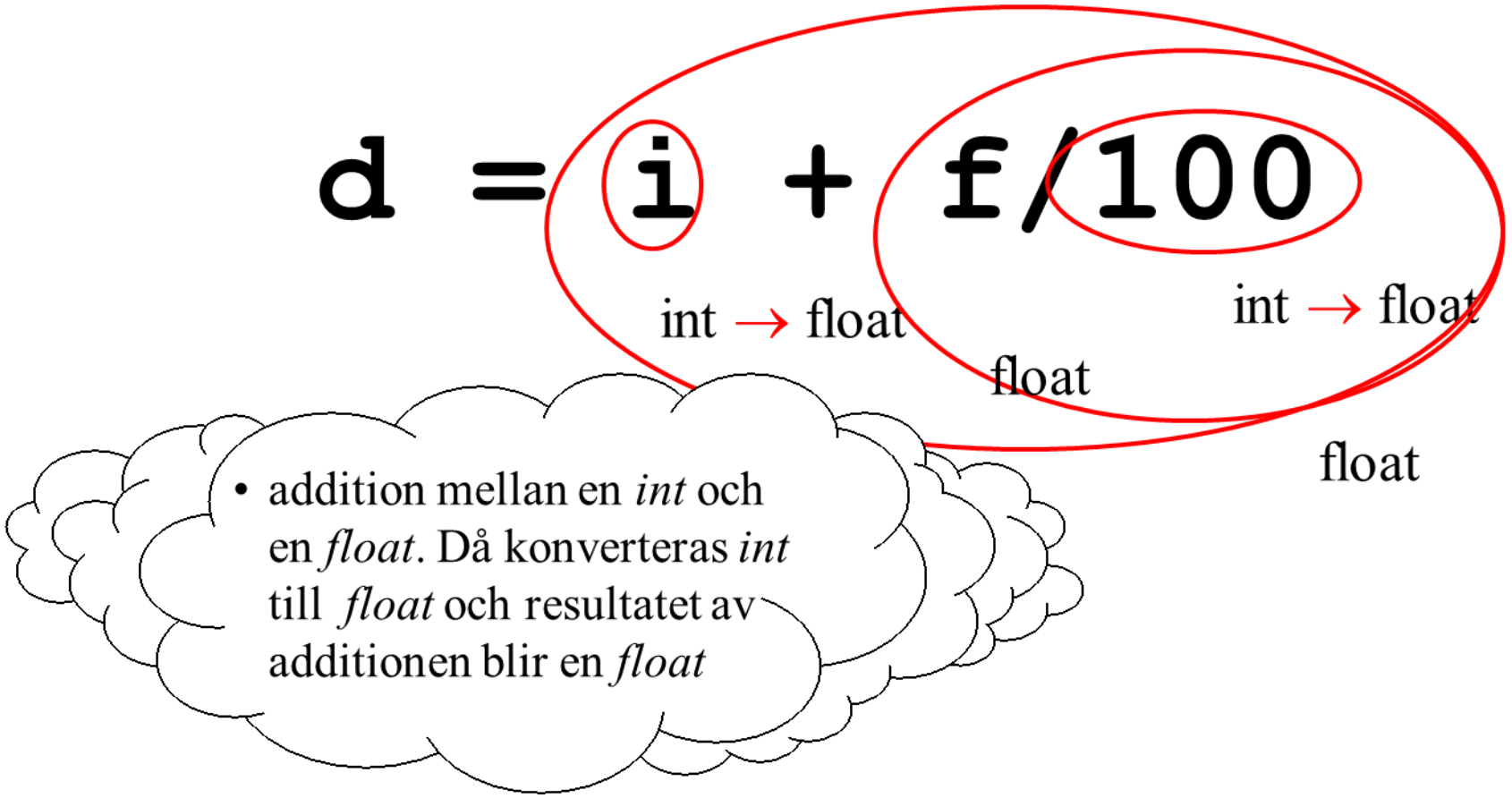
int → float

float

division mellan en *float* och en *int*. Då konverteras *int* till *float* och resultatet av divisionen blir en *float*

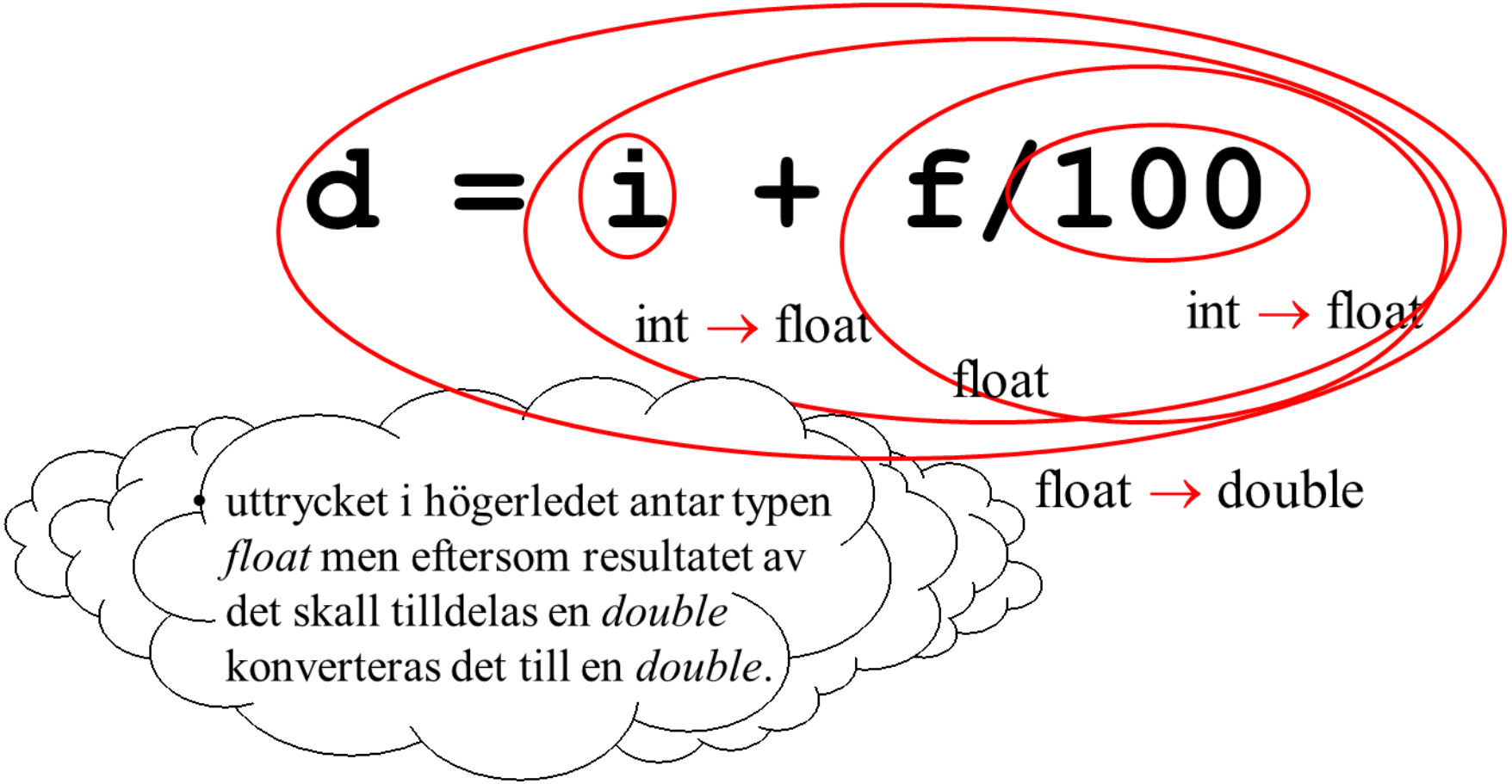
William Sandqvist william@kth.se

Typomvandling implicit (automatisk)



William Sandqvist william@kth.se

Typomvandling implicit (automatisk)



William Sandqvist william@kth.se

Typomvandling implicit (automatisk)

```
long double
double
float
unsigned long int
long int
unsigned int
int
unsigned short int
short int
unsigned char
signed char
```

högsta typ

lägsta typ

typomvandlingar där programmet får ”välja typ självt”, sker typomvandling från ”lägre” typ till ”högre” typ

William Sandqvist william@kth.se

Typomvandling

- alla typer som är av tal-typ (skalär typ) kan konverteras mellan varandra
- man bör fundera och kontrollera vad som händer vid typomvandling, följand gäller dock
  - från flyttal till heltal
    - om talet ryms (till storlek) så trunkeras det dvs **avhuggning av decimaldel** sker, ingen avrundning
  - från heltal till fyttal
    - **precisionsförlust**, det är inte säkert att flyttalsformen kan lagra alla värdesiffror

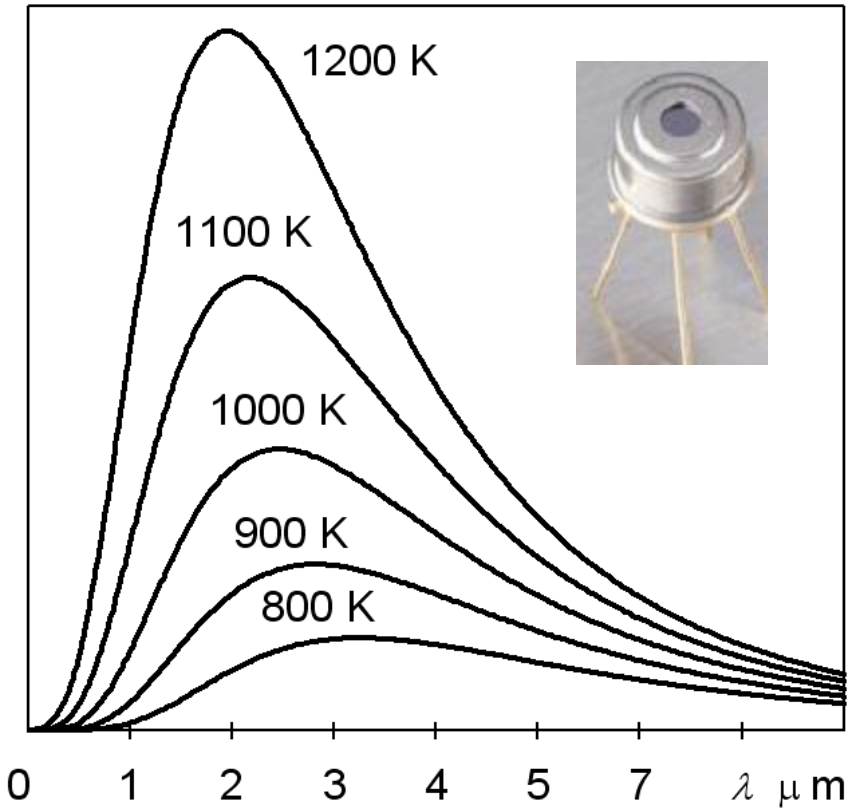
Typomvandling

explicit (av programmeraren påkallad, cast)

(*typnamn*) *uttryck*

Praktiskt exempel. En Strålningstermometer är ansluten till en 10 bitars AD-omvandlare. Mätvärdet måste justeras med hjälp av C:s matte-funktioner.

Strålningstermometer, Plank's lag



Totalstrålningens temperaturberoende.

Ytan under kurvorna är proportionell mot absoluta temperaturen T^4.

Temperaturen får man således genom att beräkna

$\sqrt[4]{\ }$ ur mätvärdet.

William Sandqvist william@kth.se

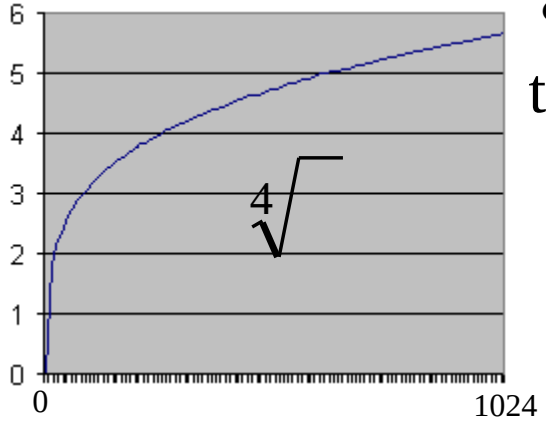
AD-omvandlare 10 bitar

$b_9 b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$

Strålningstermometer

AD-omvandlare
10 bitar
1024 steg
5 mV/steg
Upplösning ≈0,1%

Linjärisering



AD

0 … 1024

short int

double

temp *T*

0 … 6.0

Offset och skalning

-

*****

Så att 0.0° och 100.0° stämmer!

William Sandqvist william@kth.se

```
#include <math.h>
short int ad_value;
double temperature;
double offset = 3.14;
double gain = 51.4;

temperature = pow( (double) ad_value, 0.25 );
temperature -= offset;
temperature *= gain;
```

C:s matematikfunktioner

linjärisering
dra fjärde roten ur mätvärdet

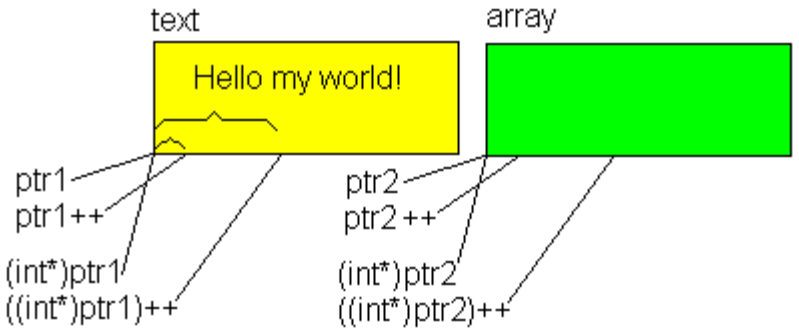
kanske justering av 0-nivå
och omskalning

Alla C:s matematik-funktioner använder flyttal med dubbel precision. Genom att typecasta visar vi vilken omvandling vi önskar.
(Vad hade C:s implicita omvandlingsregler gett för resultat denna gång?)

Typecast av pekare

Vi kan kopiera 16 bokstäver som 4 int – detta går snabbare (om datorn har 32 bitars buss).

```
#include <stdio.h>
int main(void)
{
  char text[16]="Hello my world!";
  char array[16];
  char * ptr1 = text;
  char * ptr2 = array;
  int i;
  for( i=0;i<4;i++)
   {
     *( ((int*)ptr2)++ ) = *( ((int*)ptr1)++ );
   }
  printf("%s\n",array);
  system("PAUSE");
  return 0;
}
```



Pekarna *typecastas* till int*, och då ökar steget från en Byte till 4 Byte!

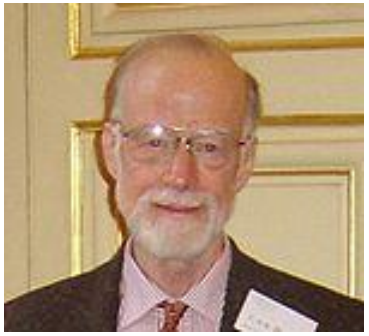
C:s biblioteksfunktioner

C:s biblioteksfunktioner är ***generellt*** skrivna och använder därför oftast **void-pekare** som parametrar och för returvärden.

Som exempel tar vi funktionen **quicksort** som storleks-sorterar element i en array för att visa hur man använder C:s biblioteksfunktioner.

Sir C.A.R Hoare

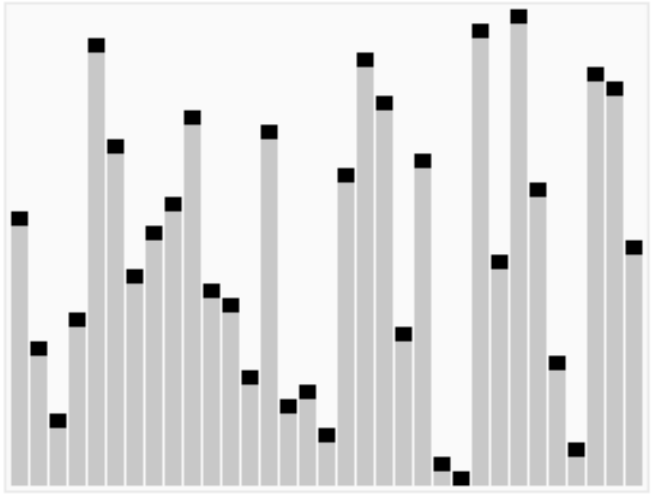
Turing award



Sortera element i en array

`qsort()`

`i stdlib.h`



Wikipedia Quicksort

William Sandqvist william@kth.se

Sorteringsfunktion Quick Sort

Alla programmeringsspråk har inbyggda sorteringsfunktioner.
C har funktionen **`qsort()`** som sorterar dataposterna i en array (eller structarray) i enlighet med en **sorteringsordning** som man själv specifierar.

Exempel, en array med fem bokstäver:

```
char alfa[5] = { 'C', 'E', 'A', 'D', 'F', 'B' };

qsort( (void*) alfa, 5, sizeof( char ), SorteringsOrdning );
```

Sorteras till: A, B, C, D, E, F om funktionen **`SorteringsOrdning()`** jämför alfabetiskt!

Pekare till arrayen

Antal element i arrayen

Storlek på elementen i Bytes

Funktionspekare, namnet på den sorteringsfunktion som skall användas

William Sandqvist william@kth.se

Sorteringsfunktionen

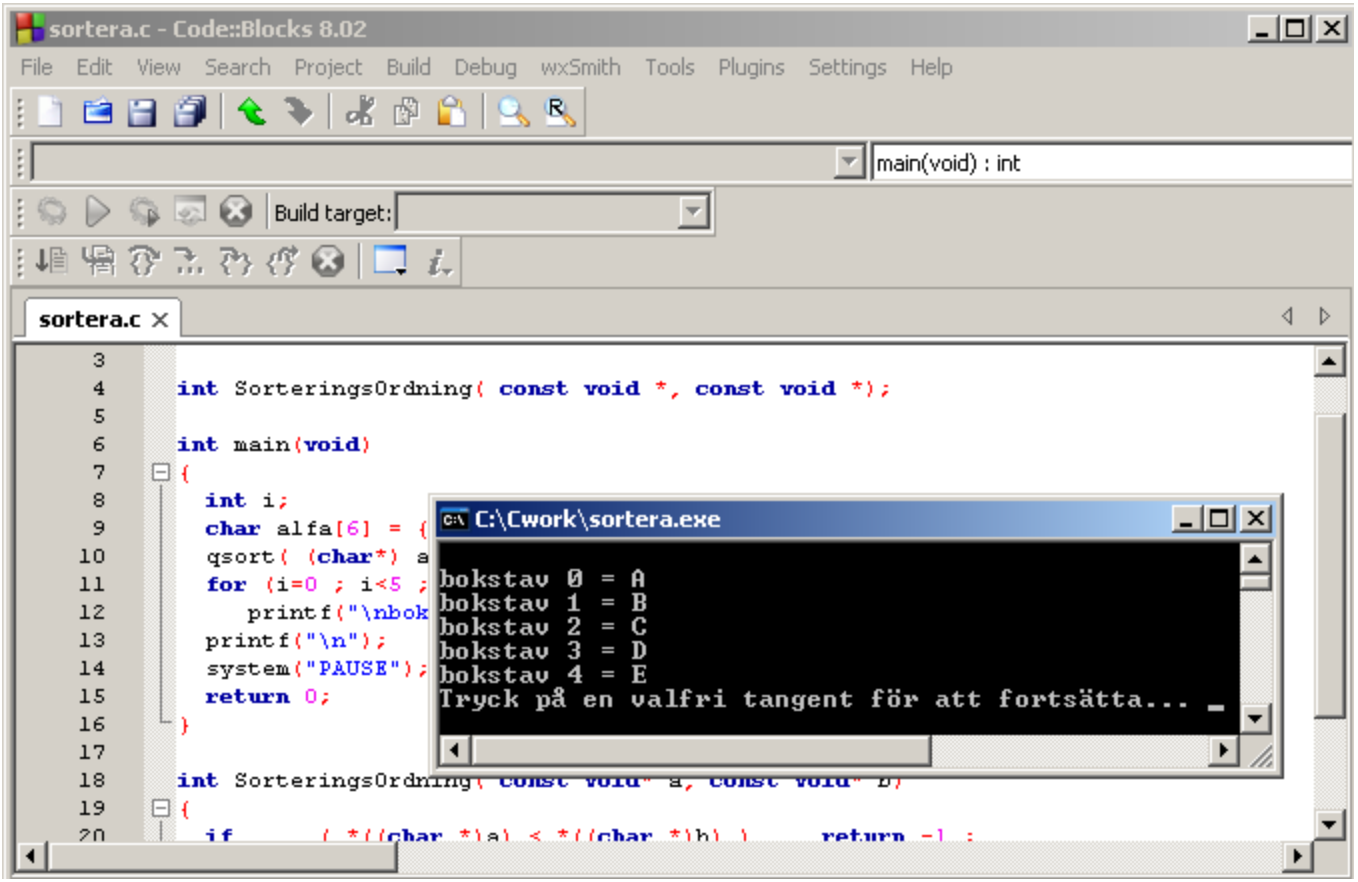
```
int Sorteringsordning( const void * a, const void * b)
{
  if ( *((char *)a) < *((char *)b) )      return -1 ;
  else if ( *((char *)a) > *((char *)b) ) return  1 ;
  else                                    return  0 ;
}
```

QSORT arbetar med void-pekare, och kan därför användas till att sortera vad som helst i arrayer. Sorteringsfunktionen, som man skriver själv och som QSORT sedan använder, skall alltid returnera -1, +1, eller 0 om parameter **b** är före, efter, eller samma som parameter **a**. På så sätt kan man sortera arrayen efter valfri egenskap.

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

int SorteringsOrdning( const void *, const void *);

int main(void)
{
  int i;
  char alfa[6] = { 'C', 'E', 'A', 'D', 'F', 'B' };

  qsort( (char*)alfa, 6, sizeof(char), SorteringsOrdning );

  for (i=0 ; i<5 ; i++)  // now in order?
     printf("\nbokstav %d = %c",i, alfa[i]);
  printf("\n");
  system("PAUSE");
  return 0;
}

int SorteringsOrdning( const void* a, const void* b)
{
  if      ( *((char*)a) < *((char*)b) )     return -1 ;
  else if ( *((char*)a) > *((char*)b) )     return  1 ;
  else                                      return  0 ;
}
```

Inte i alfabetisk ordning!

Nu alfabetiskt sorterat?



William Sandqvist william@kth.se

Ex. Medianfilter

Ett **medianfilter** tar hela tiden fram det till storleken **”mittersta”** värdet av löpande mätvärden. Man behöver då kunna storlekssortera mätvärden – kanske med **quick-sort**! Brus-störningar undertrycks extremt bra med ett sådant filter.

Ex. varje pixel har ersatts med **medianvärdet** av de pixlar som finns inuti en cirkel (eller kvadrat) med pixlet som centrum.

William Sandqvist william@kth.se

Egna datatyper

Eftersom storleken på C:s datatyper kan variera är det ganska vanligt att programmerare definierar om de datatyper som finns till egna namn – med mer exakt betydelse.

```
typedef unsigned long ULONG;
```

Från och med nu så kan en variabel vara av typen:

```
ULONG tal = 1234567890;
```

ALTERA har egna datatyper, Windows har egna …

struct elephant

Från Datortekniken ….

```
typedef struct
{
   float left;
   float right;
} oron ;

typedef struct
{
  char namn[13];
  int vikt;
  oron oronarea;
} elephant ;
```

Från och med nu finns det en datapost av typen

elephant

William Sandqvist william@kth.se

Delminnen

Så här når vi delminnen:

```
elephant dumbo;
elephant * pek = &dumbo;
dumbo.oron.right = 0.25;
pek->oron.left = 0.36;
```

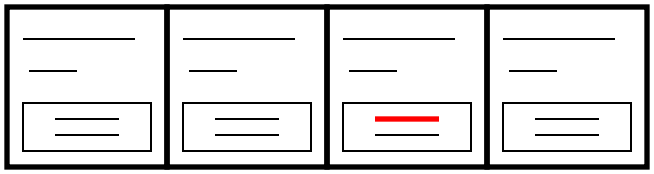
`.` punktoperator

`->` piloperator

Vektor `circus[]`

deklarerar en vektor, **`circus[]`**, med plats för fyra elefanter.

```
elephant circus[4];
```

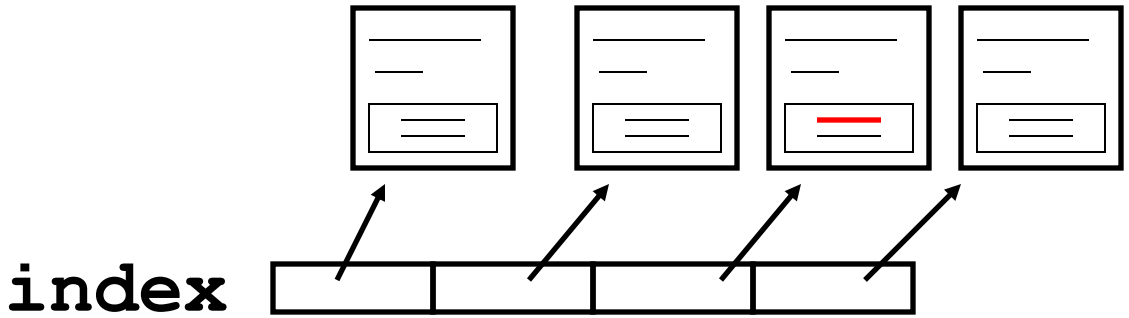
`circus` 

```
circus[2].oron.left = 0.56;
```

Indexarray **`index[]`**

deklarerar en vektor, **`index[]`**, med plats för fyra elefant-pekare.

```
elephant * index[4];
```

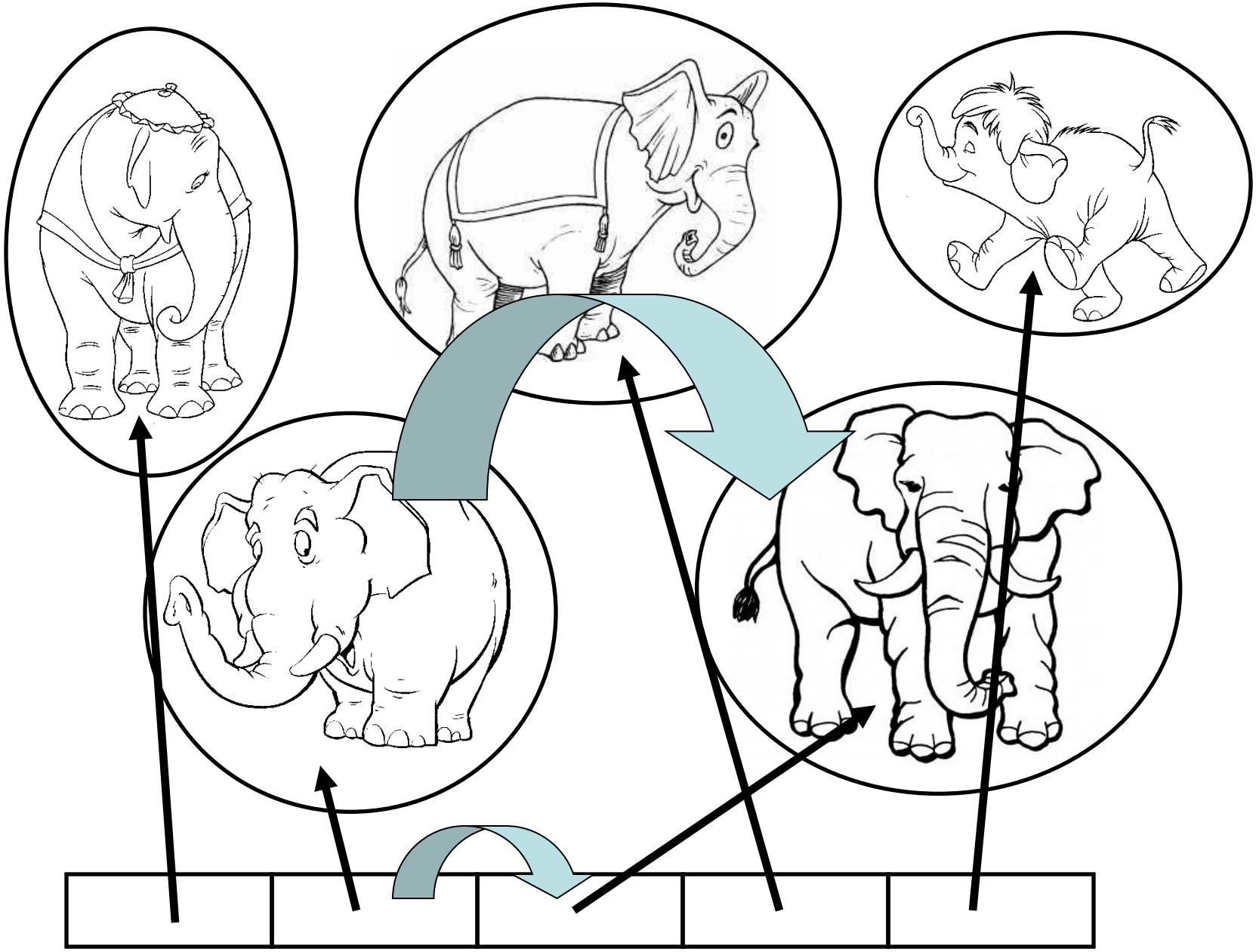



```
index[2]-> oron.left = 0.56;
```

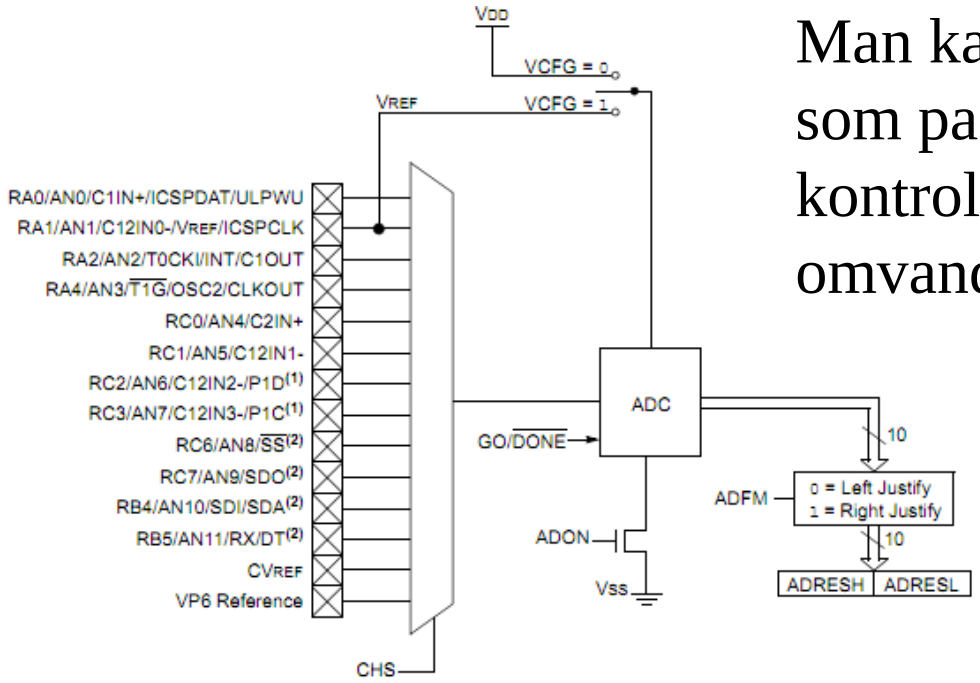
Hur sorterar man elefanter?

Genom att sortera pekarna – inte gärna genom att flytta på elefanterna!

William Sandqvist william@kth.se



Bitfield



Man kan skapa en datatyp som passar tex. ett kontrollregister för en AD-omvandlare.

REGISTER 9-1: ADCON0: A/D CONTROL REGISTER 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	VCFG	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

William Sandqvist william@kth.se

Bitfield

REGISTER 9-1: ADCON0: A/D CONTROL REGISTER 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	VCFG	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

```
typedef struct
{
  unsigned ADFM :1;
  unsigned VCFG :1;
  unsigned CHS  :4;
  unsigned GO   :1;
  unsigned ADON :1;
 } ADCON0 ;
```

```
ADCON0 tempinit;
tempinit.CHS = 9;
tempinit.GO = 0;
tempinit.ADFM = 1;
tempinit.VCFG = 0;
tempinit.ADON = 1;
```

Tänkbar användning:

```
void AD_init( ADCON0 init );
```

William Sandqvist william@kth.se

`volatile` ?

Periferienheter, I/O, ansluts ofta till en CPU som om dom vore minneskretsar (fast med bara ett fåtal ”minnesceller”). Ex. en realtidsklock-krets – håller reda på tid och datum. Den styrs/avläses från 8 inbyggda register.

Eftersom I/O-enheter *inte* är riktiga minnen – det kan verka som om innehållet kan ändras ”av sig självt” – så kan man vid programmeringen av processorn behöva ”hjälpa” kompilatorn att förstå detta genom att deklarera dem som **`volatile`**

(= flyktiga) i sina C-program.

Har till exempel processorn ett **cache-minne** är det viktigt att man läser klock-kretsen direkt och inte några *äldre* ”cachade” tider!

William Sandqvist william@kth.se

`const` ?

`const` innebär att en variabel inte går att modifiera under körningen (ingen tilldelning kan ske). Variabeln initieras därför när den definieras.
En pekare som är **`const *`** kan peka på olika adresser – och man kan då läsa på dessa adresser men inte skriva där.

`const` deklarationen gör att kompilatorn kan kontrollera att man inte uttryckligen ändrar en variabel någonstans i programmet.
En sådan variabel kan placeras i ett läsminne ROM.

`#pragma` ?

Några exempel på användningen av pragma – här gäller det en liten PIC-processor.

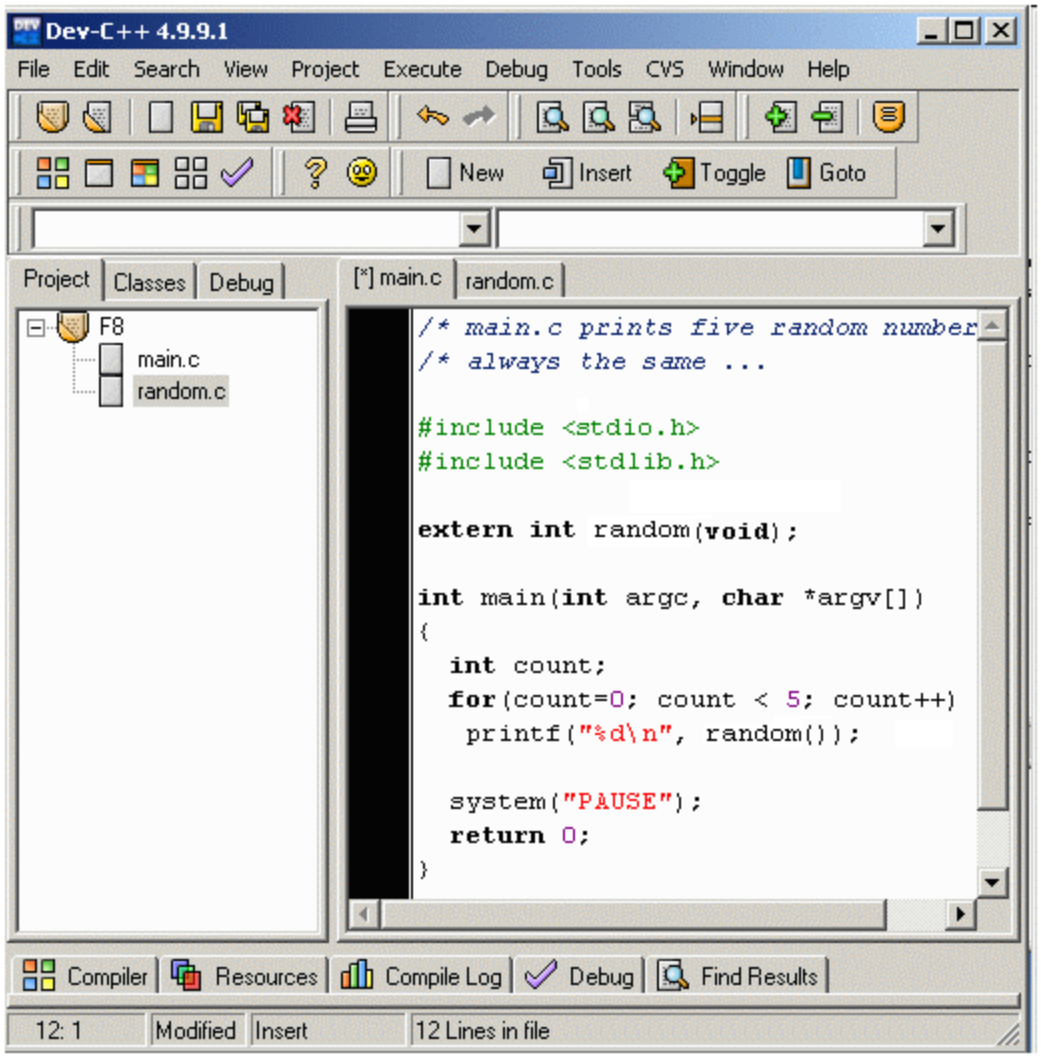
```
#pragma config |= 0x3fb0

#pragma bit lightdiode @ PORTB.0
```

En pragmatisk person är en som låter sitt handlande styras av vad som är praktiskt hellre än vad reglerna föreskriver.

Med Preprocessorkommandot **`#pragma config |= 0x3ff1`** förs information om vilka inställningar som man ska använda vid själva Krets-programmeringen över till krets-programmeraren.
Preprocessorkommandot **`#pragma`** betyder att man gjort en sådan pragmatisk avvikelse från ANSI-C standarden.
`#pragma bit lightdiode @ PORTB.0` betyder att man infört en helt ny datatyp, bit, bitvariabel (som kan rymma 1 eller 0). Vi ger därefter bitvariabeln namnet **`lightdiode`** och tecknet `@` betyder att variabeln har en fast adress (kompilatorn får inte placera den någon annanstans).

William Sandqvist william@kth.se

Användningen av `static`



William Sandqvist william@kth.se

Lagringsklasser och deklarationsområde

Exempel på användning av lagringsklassen static. Portabla slumptal (Pseudorandom numbers). Exemplet från **C Primer Plus**, *Stephen Prata*, ISBN 1-57169-161-8.
Skapa ett projekt i Dev-C++ med två filer `main.c` och `random.c`.
Kompilera och provkör.

/* main.c prints five random numbers */

```
#include <stdio.h>
#include <stdlib.h>
extern int random(void);

int main(int argc, char *argv[])
{
  int count;
  for(count=0; count < 5; count++)
    printf("%d\n", random());
  system("PAUSE"); return 0;
}
```

Slumpfunktionen

```
/* random.c produces random numbers */
/* uses ANSI C portable algorithm   */

static unsigned long int next =1;  /* the seed */

int random(void)
{
  /* magic formula to generate pseudorandom number */
  next = next * 1103515245 + 12345;
  return (unsigned int) (next/65536) % 32768;
}
```

Slumpfunktionen, problem

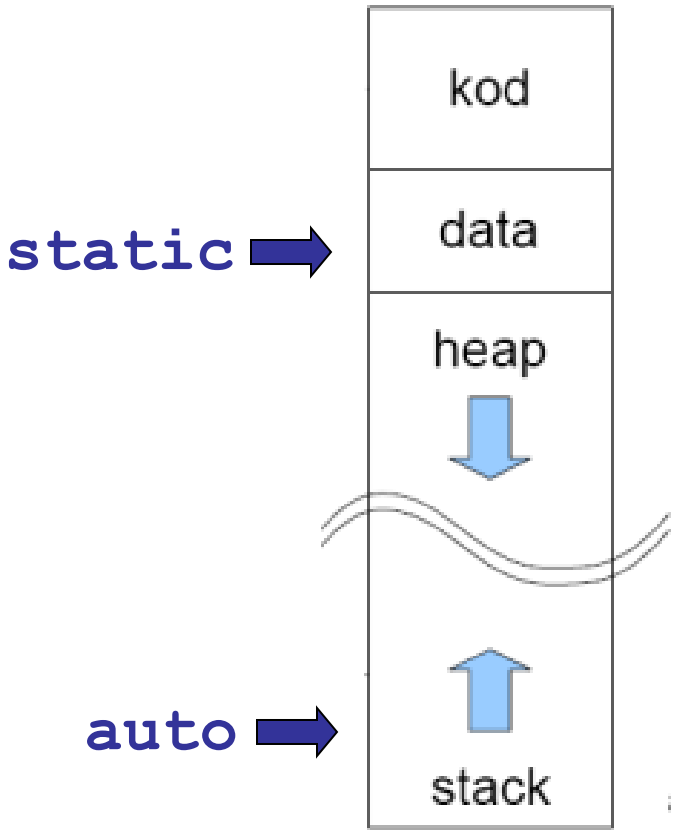
Vad skulle hända om next *inte* deklareras som static?

```
/* random.c produces random numbers */
/* uses ANSI C portable algorithm   */

int random(void)
{
  unsigned long int next =1;     /* the seed */

  /* magic formula to generate pseudorandom number */
  next = next * 1103515245 + 12345;
  return (unsigned int) (next/65536) % 32768;
}
```

Ooops!

William Sandqvist william@kth.se

static



En variabel som är **`static`** ”lever” för evigt (som en global) variabel, men den är bara känd i det block där den är definierad och kan bara nås där.

En lokal variabel i en funktion (**`auto`**) ”lever” bara så länge som funktionen körs.

William Sandqvist william@kth.se

Slumpfunktionen, mer problem

```
/* random.c produces random numbers */
/* uses ANSI C portable algorithm   */

static unsigned long int next =1;  /* the seed */
int random(void)
{
  /* magic formula to generate pseudorandom number */
  next = next * 1103515245 + 12345;
  return (unsigned int) (next/65536) % 32768;
}
```

Måndag

Tisdag

Onsdag

Samma slumptalsserie vid varje körning … (lösning! byt seed/Startvärde mellan körningarna)

Rapport om program för ett inbyggt system

Man dokumenterar:

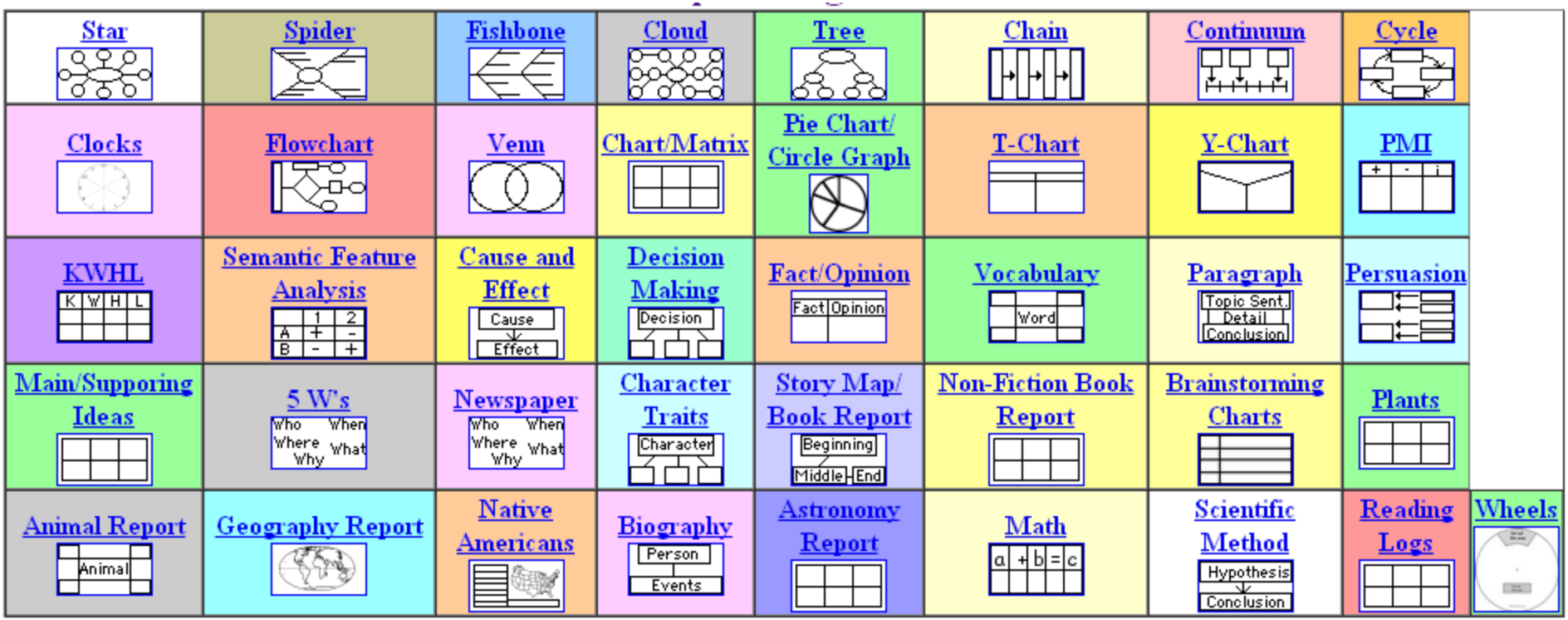
- Systemets uppbyggnad
- Programmets omfattning
- Programmets uppbyggnad
- Hur programmet/funktionen testats

Tekniska rapporter innehåller ”figurer” och text i samverkan. Figurerna är ofta olika typer av diagram.

William Sandqvist william@kth.se

Graphic Organizers

En sida med diagram för *grundskoleelever*.

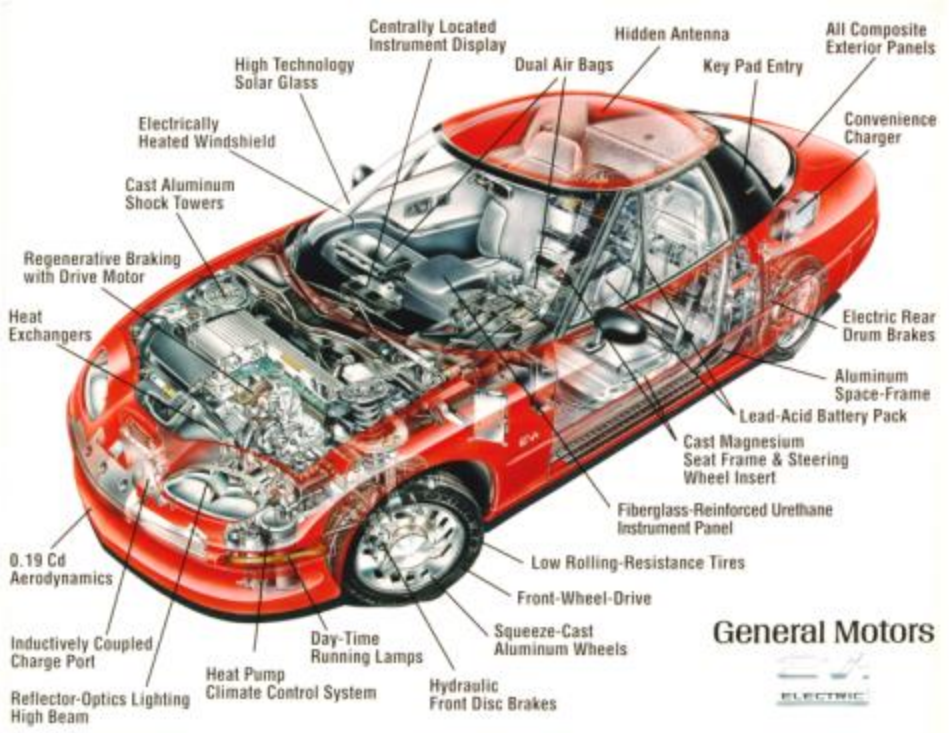


Det kan vara enklare att förklara något med hjälp av ett diagram än utan. **http://www.enchantedlearning.com/graphicorganizers/**

William Sandqvist william@kth.se

Sammanställningsbild

En inbyggnadsprocessor används inbyggd i en produkt. Det går nog knappast att beskriva processorns dator-program utan att man först åtminstone översiktligt förklarat **produkten**!

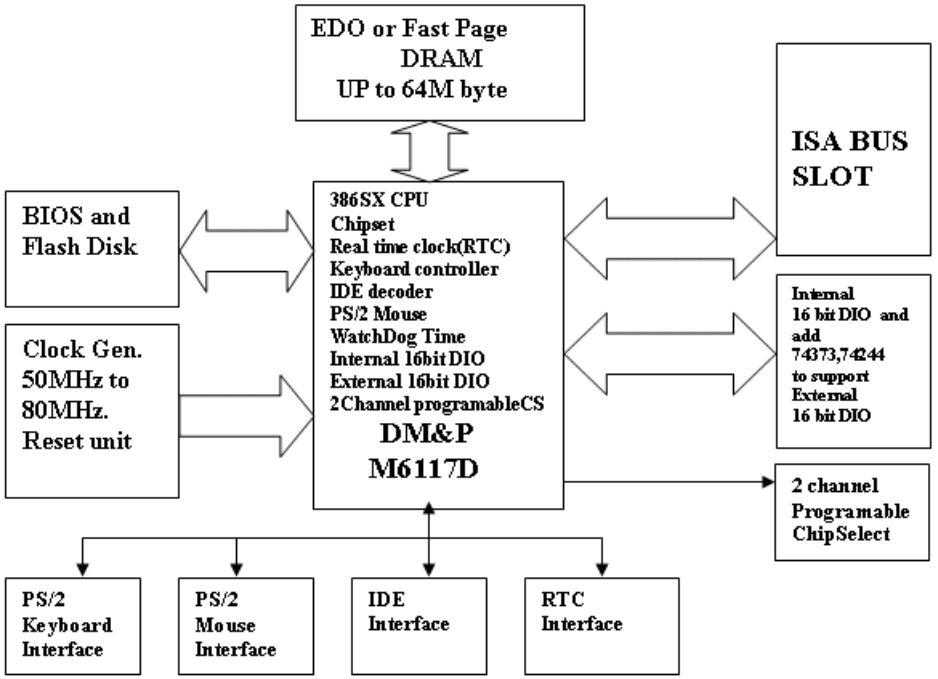


En **sammanställningsbild** definierar vad de olika delarna kallas för – man använder sedan exakt de namnen konsekvent genom hela rapporten när man behöver hänvisa till produkten.

William Sandqvist william@kth.se

Blockdiagram

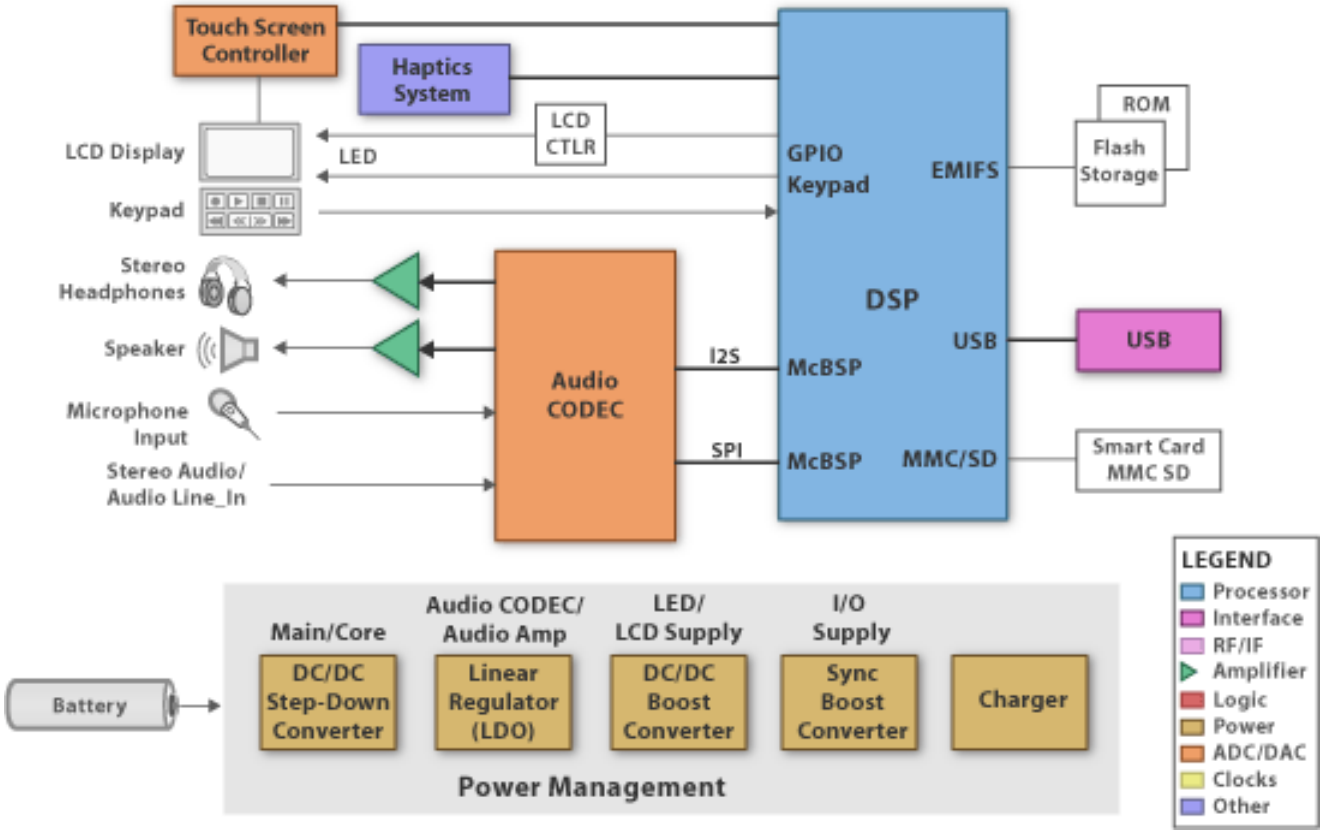
Ett blockdiagram är ett diagram över ett system där de principiella delarna, eller funktionerna, representeras av block sammanbundna med pilar som visar blockens relationer.



Det krävs inga avancerade hjälpmedel för att rita ett blockdiagram. Blockdiagrammet gäller produktens delar. Andra diagramtyper används för att beskriva programmets struktur.

William Sandqvist william@kth.se

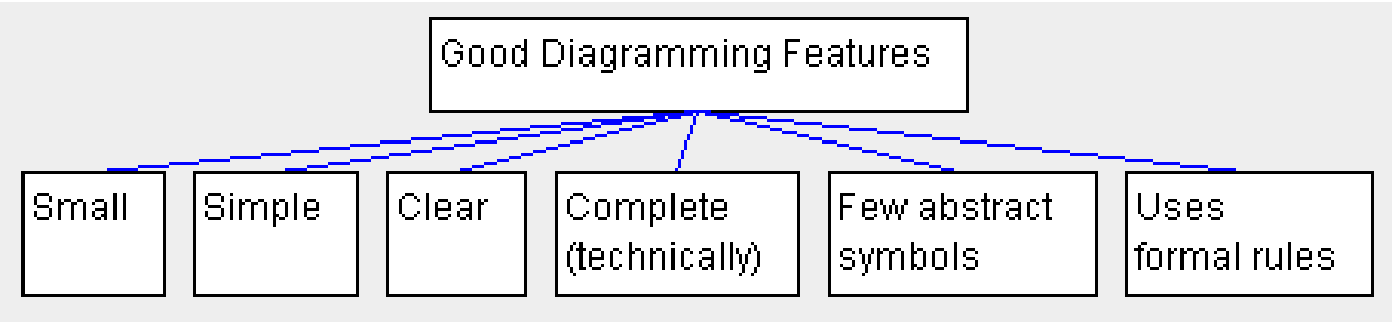
Illustrerat blockdiagram



Vänder man sig till läsare utanför gruppen av tekniker händer det ofta att blocken är färglagda och illustrerade …

William Sandqvist william@kth.se

Bra diagramegenskaper

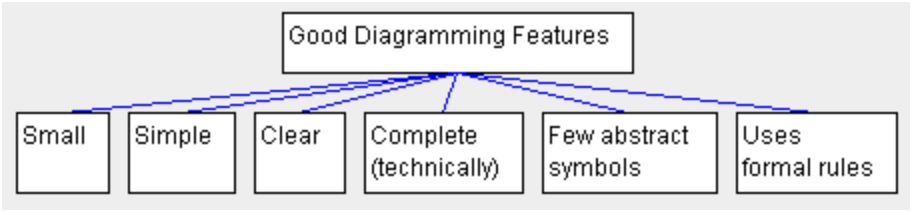


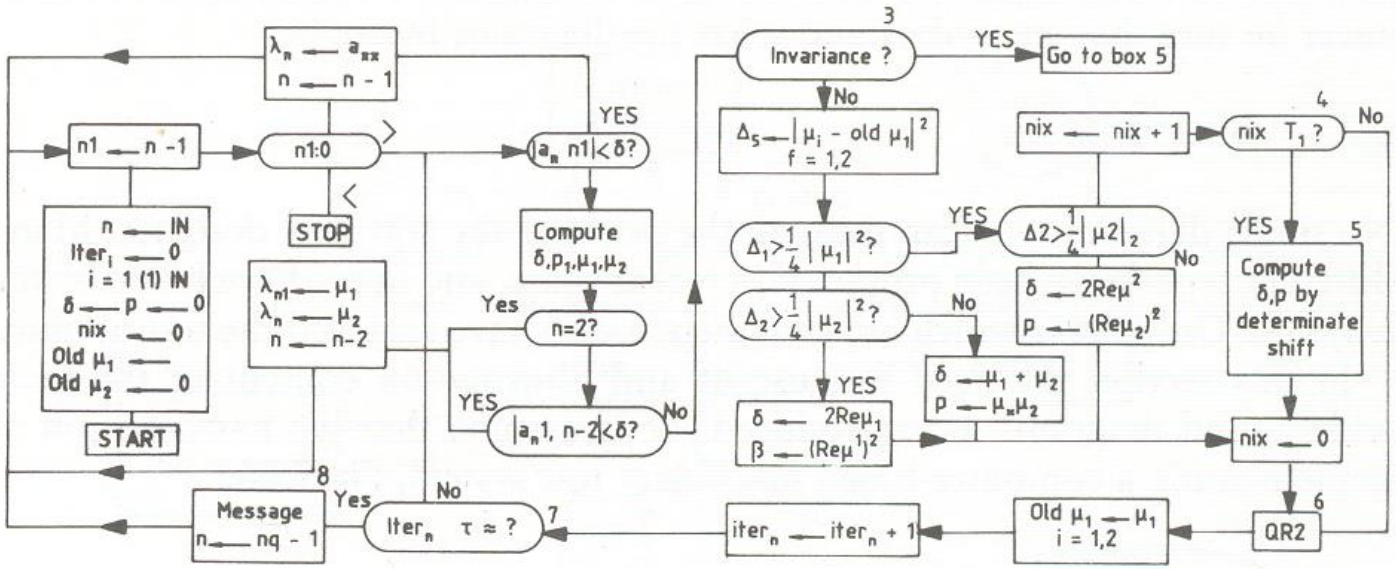
Ett bra diagram ska vara litet, enkelt, klart, innehålla få abstrakta symboler, och följa formella regler.

Ett bra diagram är till god hjälp när man ska förklara systemet för andra.

Exempel på ett diagram som bryter mot alla reglerna!

Bra diagram-egenskaper



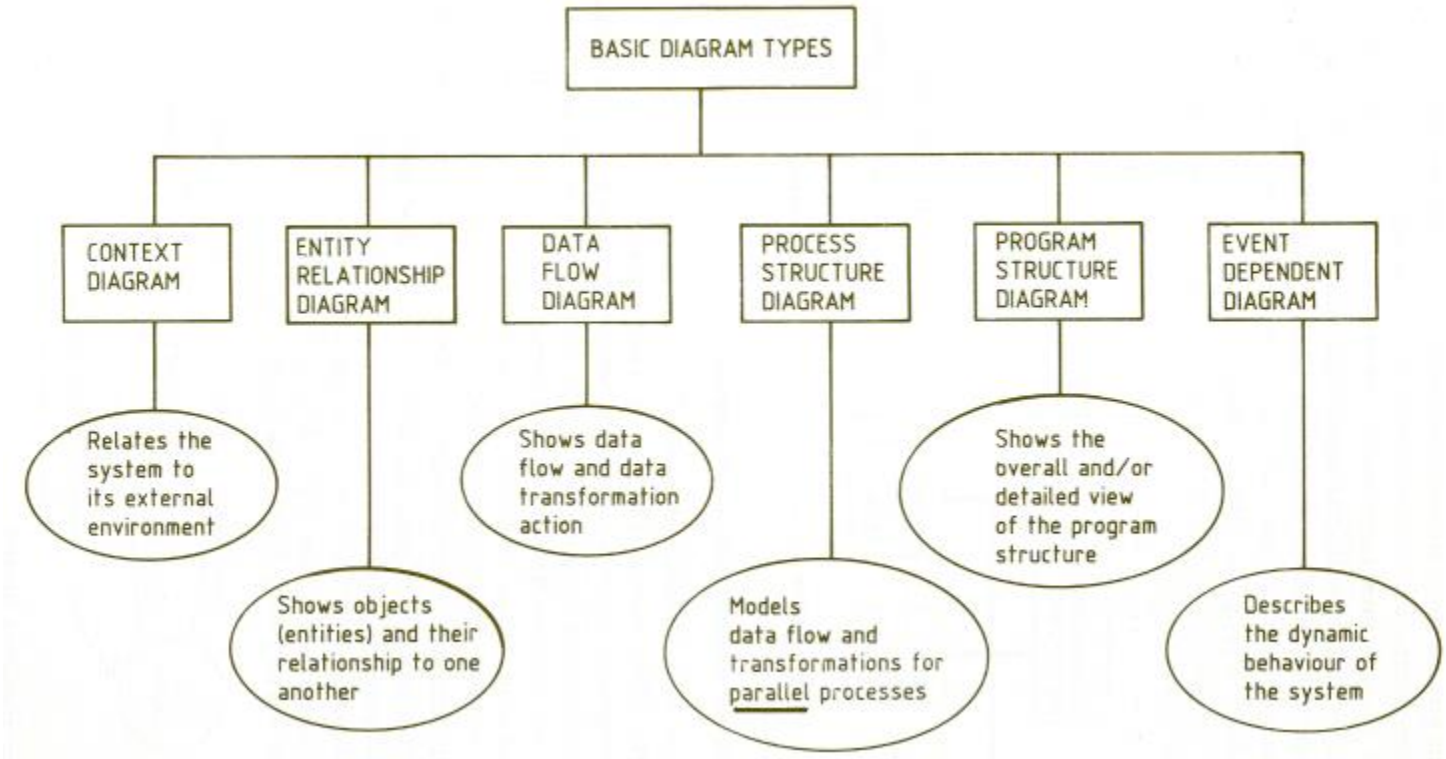


William Sandqvist william@kth.se

Diagrammet blir bara till hjälp om Du anstränger dig …

William Sandqvist william@kth.se

Diagramtyper för programkod

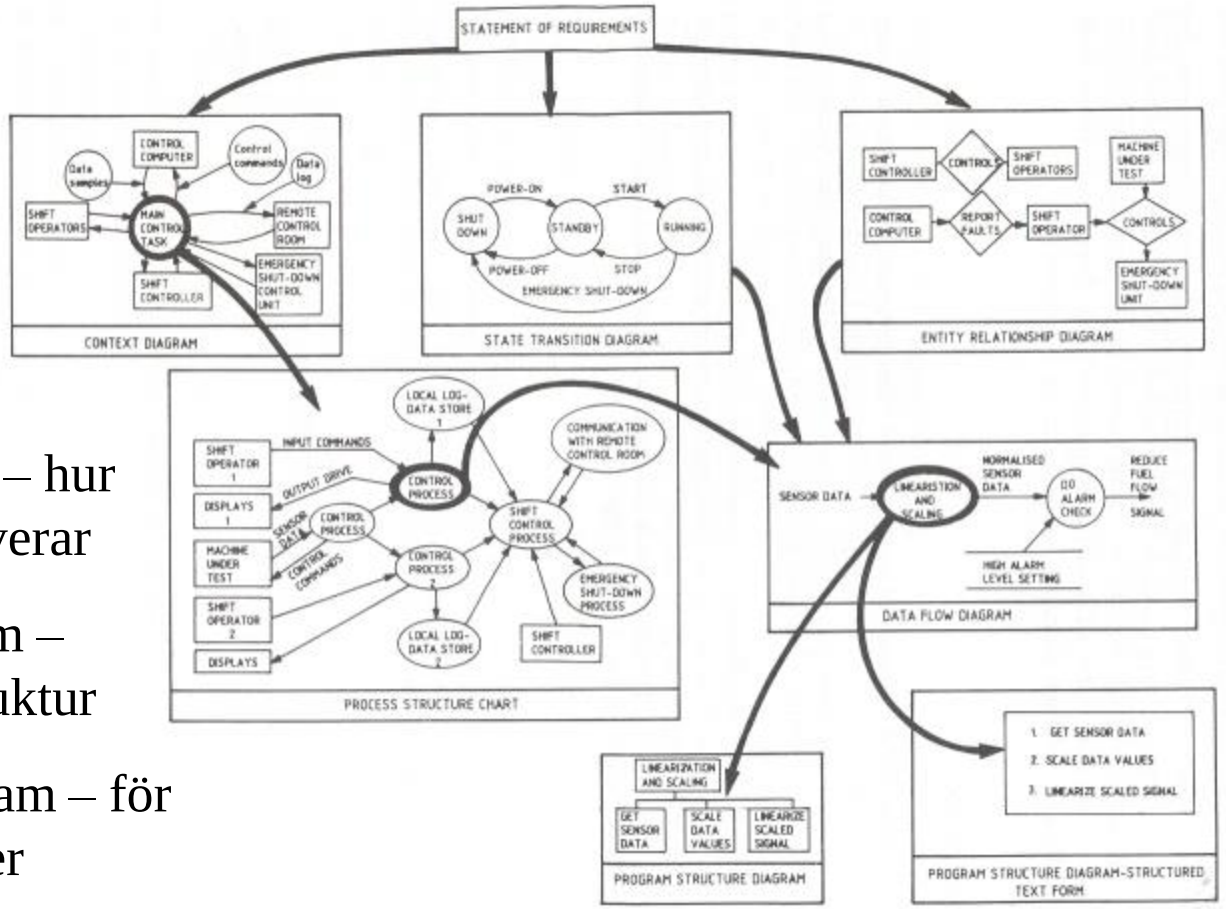


Det finns många diagramtyper för att täcka *olika aspekter* av ett datorprogram.

Diagramtyper för programkod

Tre vanliga diagramtyper:

- Flödesdiagram – hur processorn exekverar
- Strukturdiagram – programmets struktur
- Tillståndsdiagram – för tillståndsmaskiner

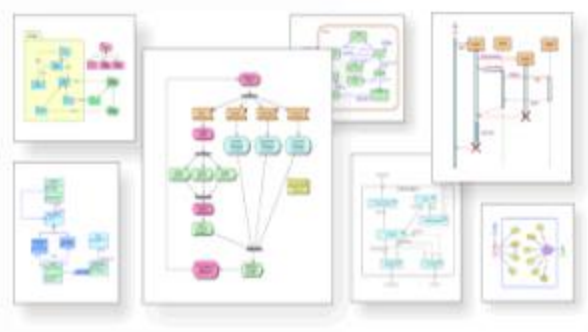


Pilarna visar hur flera diagramtyper samverkar och behövs för att ge en helhetsbild.

UML är för omfattande för denna kurs …

UML

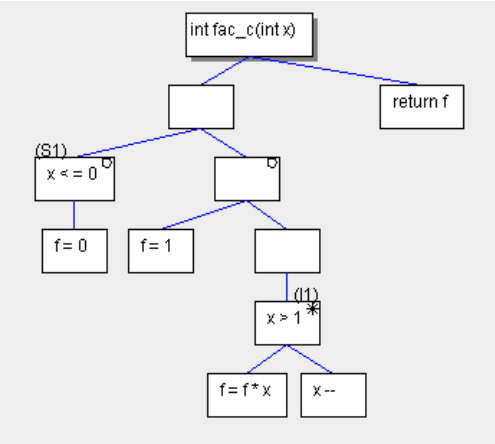
Unified Modeling Language (**UML**) är ett objektorienterat generellt språk för modellering av alla typer av system.



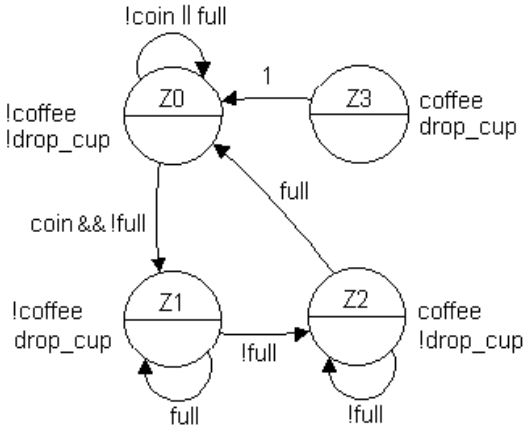
UML – innehåller en mängd olika diagramtyper.

Språket används främst inom programvarukonstruktion men är även användbart inom andra områden som exempelvis modellering av affärsprocesser med mera. Genom att skapa en modell av systemet som skall konstrueras blir det enklare att förstå och bygga det.

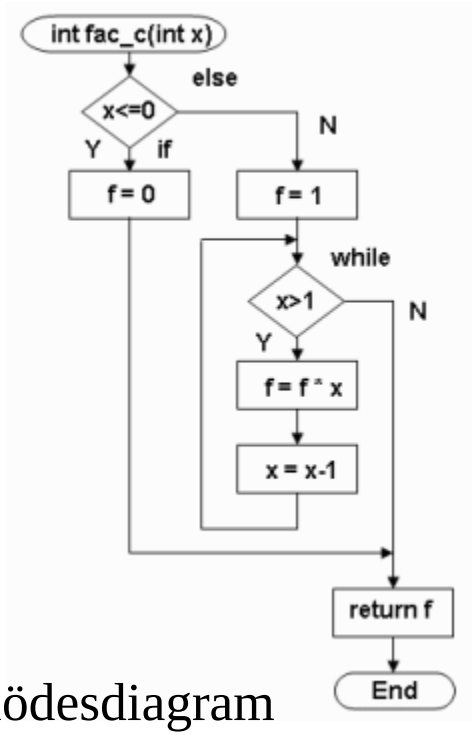
Vi använder …



Strukturdiagram
– programmets struktur



Tillståndsdiagram
– för tillståndsmaskiner



Flödesdiagram
– hur processorn exekverar

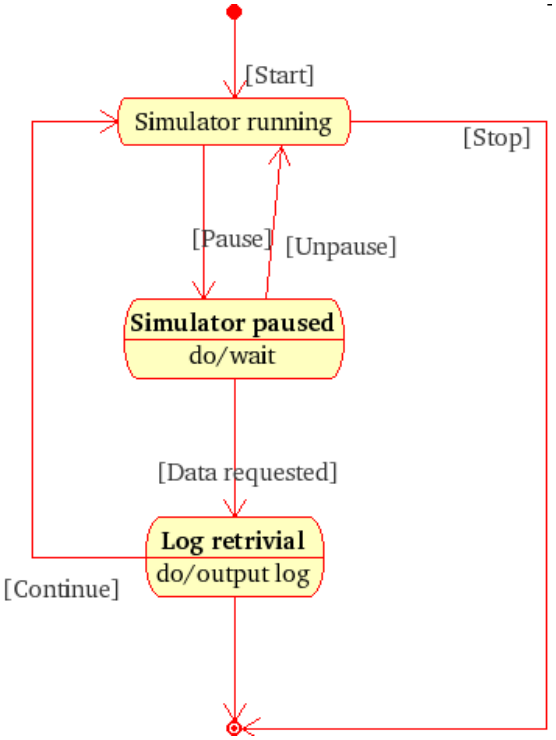
William Sandqvist william@kth.se

Programmering efter tillståndsdiagram

En mycket vanlig teknik vid programmering av inbyggnadsprocessorer är att använda ”tillstånd” och ”tillståndsdiagram”.

Idén är lånad från Digitalteknikens ”automater”.

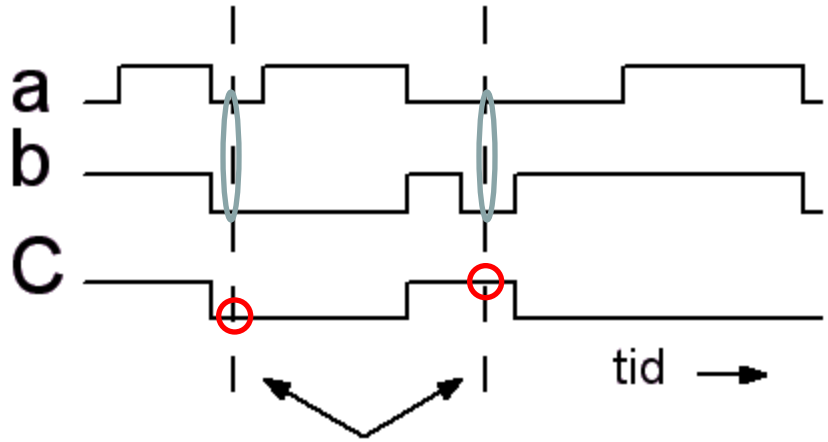
Låt oss därför se kort på några av digitalteknikens sekvenskretsar och automater.

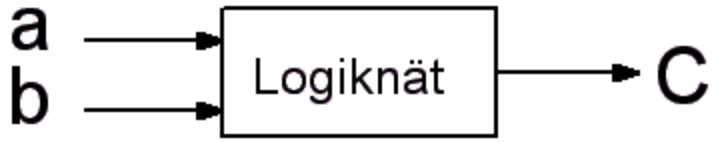


UML-tillståndsdiagram

William Sandqvist william@kth.se

Sekvensnät





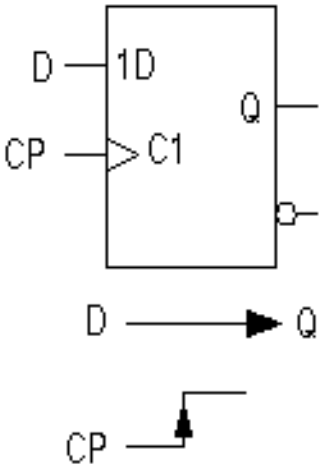
Om en och samma insignal kan ge upphov till olika utsignal, är logiknätet ett sekvensnät.
Det måste då ha ett ***inre minne*** som gör att utsignalen påverkas av både nuvarande och föregående insignaler!

William Sandqvist william@kth.se

Repetition - Klockad vippa

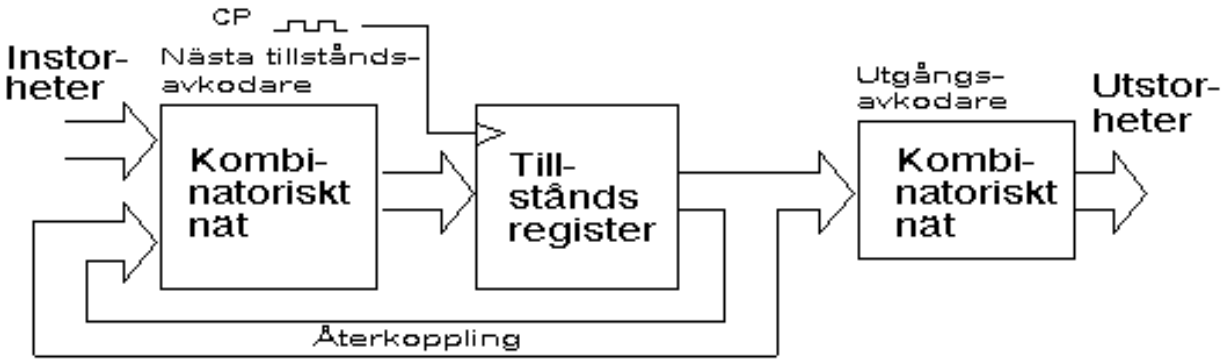
Inom digitaltekniken brukar man skilja på enkla låskretsar (latch) och klockade vippor (flip-flop).
De enkla låskretsarnas begränsning ligger i att man *inte* kan mata in ett nytt värde till ingången *samtidigt* som man läser av utgångsvärdet. De elektroniska kretsarnas snabbhet har gjort det nödvändigt att utveckla mer sofistikerade kretsar.

Flanktriggad D-vippa. D-ingången är dataingång, C-ingången är klockpulsingång, därav beteckningen CP. Styringången C har ett flanktriggningstecken, en triangel. När C-ingången nås av en **positiv flank**, det vill säga under den korta tid då C går från "0" till "1”, kopieras D-ingångens värde till utgången Q. Utgångsvärdet är sedan låst tills det inkommer en ny flank på klockpulsingången.



William Sandqvist william@kth.se

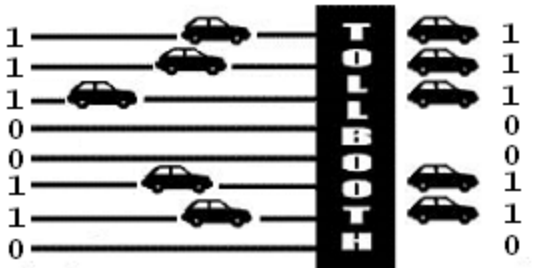
Moore-automat

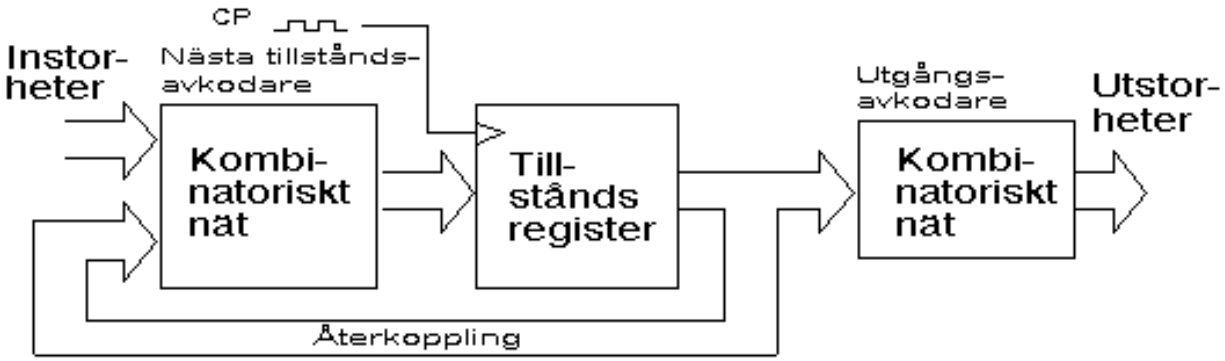


En vanlig typ av sekvensnät är Moore-automaten. Nätets ”minnesfunktion” sitter i ett **Tillståndsregister** uppbyggt av D-vippor. Ett logiknät – **Nästa tillståndsavkodaren** – förutsäger det kommande tillståndet utifrån det nuvarande tillståndet och insignalerna. När klockpulsen CP kommer hamnar man i det nya tillståndet.

Tillståndsregistrets D-vippor

Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).

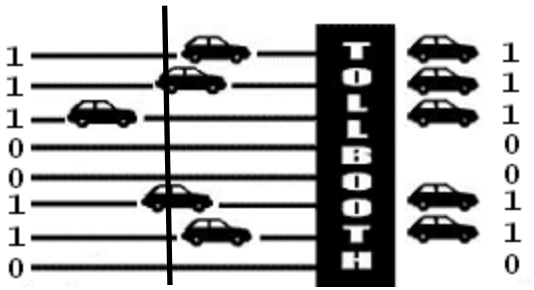


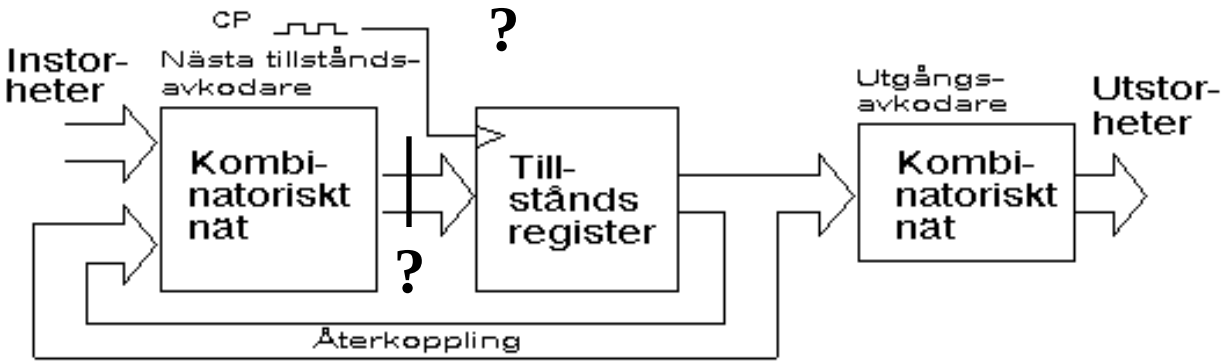


William Sandqvist william@kth.se

Tillståndsregistrets D-vippor

Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).

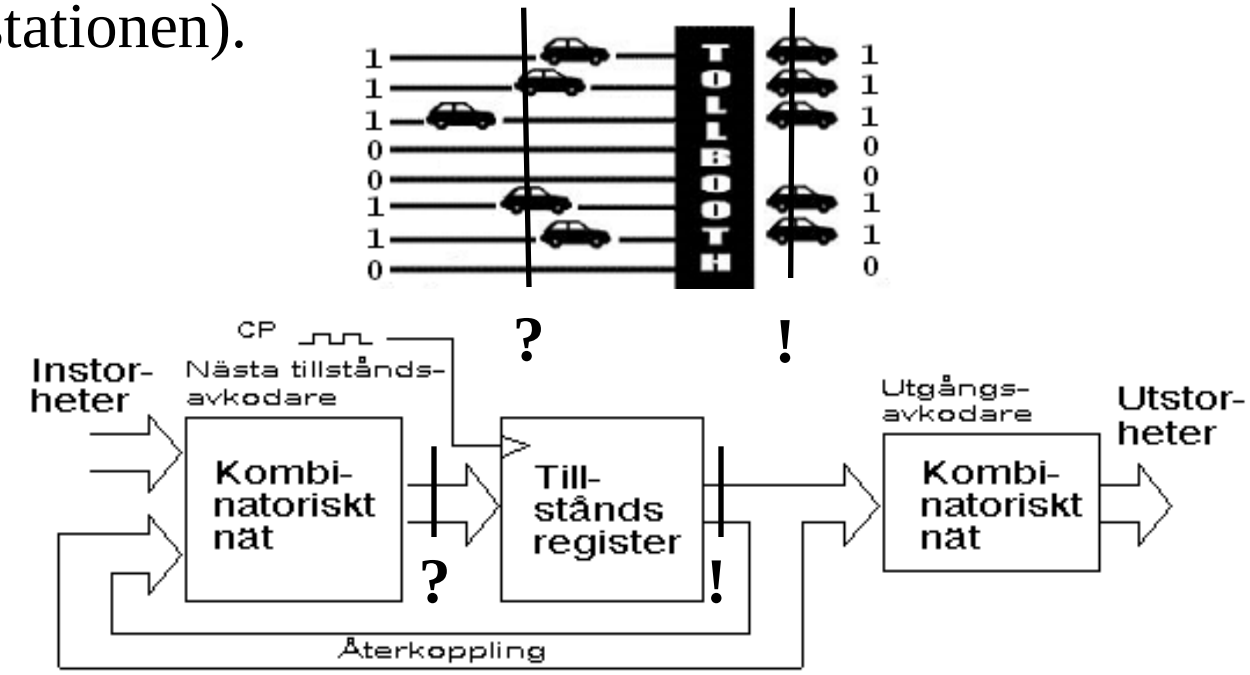




William Sandqvist william@kth.se

Tillståndsregistrets D-vippor

Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).



William Sandqvist william@kth.se

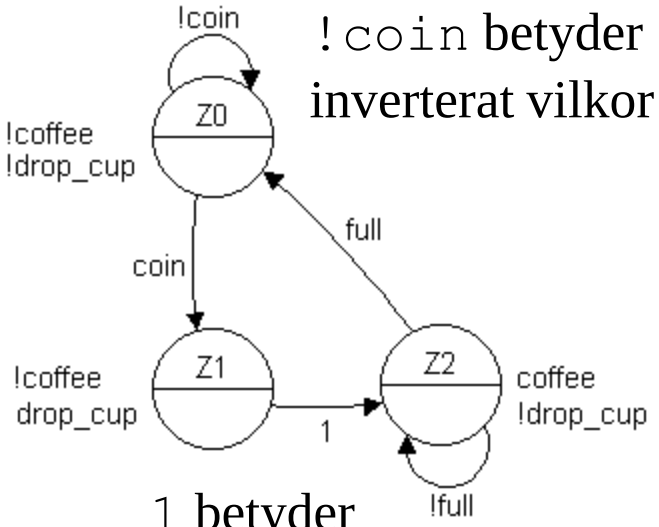
Ex. Kaffeautomaten

William Sandqvist william@kth.se

Exempel: En kaffeautomat

Nästa tillståndsavkodaren ska förutsäga nästa tillstånd – ungefär som ***oraklet i Delfi*** förutsäger framtiden!
Uppgiften är *inte omöjlig*, det finns en systematisk metod som steg för steg leder till målet.

Första steget är att rita ett **Tillstånds-diagram**.
Cirklarna **Z0 Z1** och **Z2** är tillstånden.
Pilarna är möjliga övergångar mellan tillstånden och texten vid pilarna är vilkoret för övergången.
Bredvid cirklarna står vad som ska hända i tillstånden.



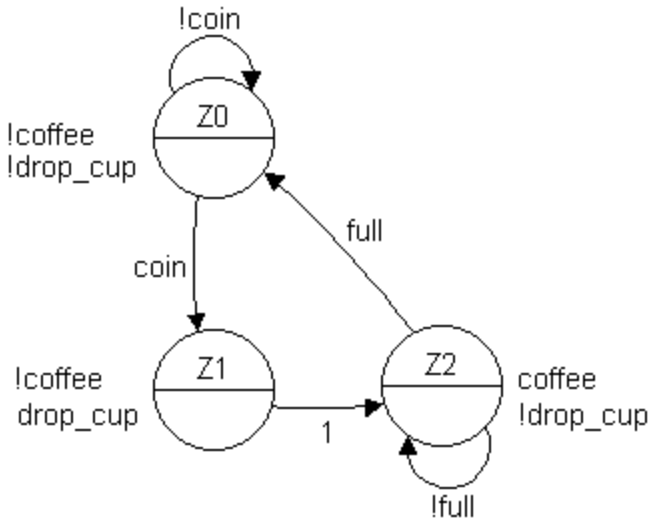
`!coin` betyder inverterat vilkor

`1` betyder alltid

Tillståndsdiagrammet, insignaler

Kaffeautomaten har två *insignaler*.

`coin` från myntinkastet som anger att en pollett passerat en fotocell där.



`full` från en givare som ”övervakar" plastmuggen under fyllningen.

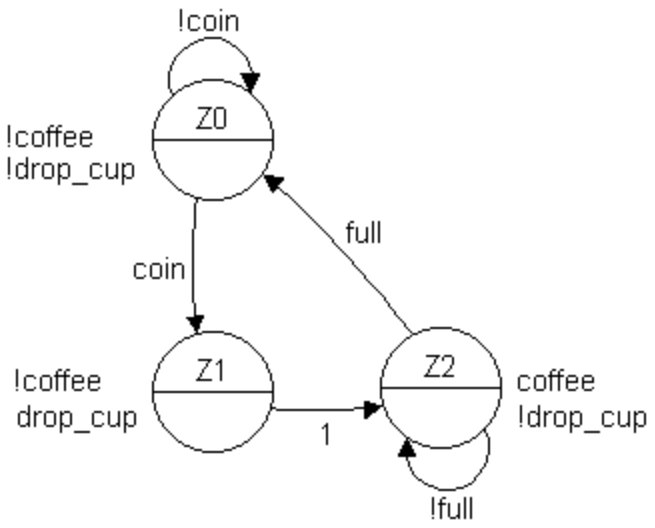
`coin` = **1** när myntet passerar. **`full`** = **1** när muggen blir full.

Tillståndsdiagrammet, utsignaler

Kaffeautomaten har två utsignaler.

`drop_cup` till en matarenhet för plastmuggar.

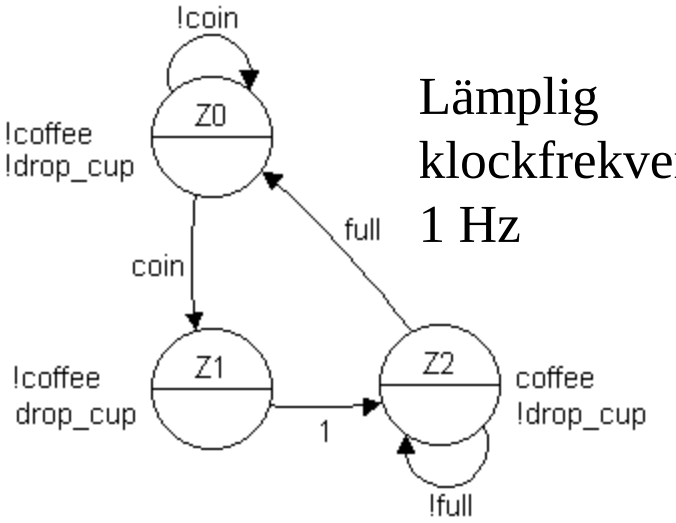
`coffee` till en magnetventil för påfyllning av kaffet.



Mataren matar fram en mugg var gång **`drop_cup`** blir "1", och kaffe fylls på så länge som **`coffee`** = **1**.

Tillståndsdiagrammet, övergångar

I första tillståndet, **Z0**, gäller för utsignalerna att man inte släpper någon kopp (**`!drop_cup`**), och inte häller i någon dryck (**`!coffee`**) förrän man fått betalning (**`coin`**).

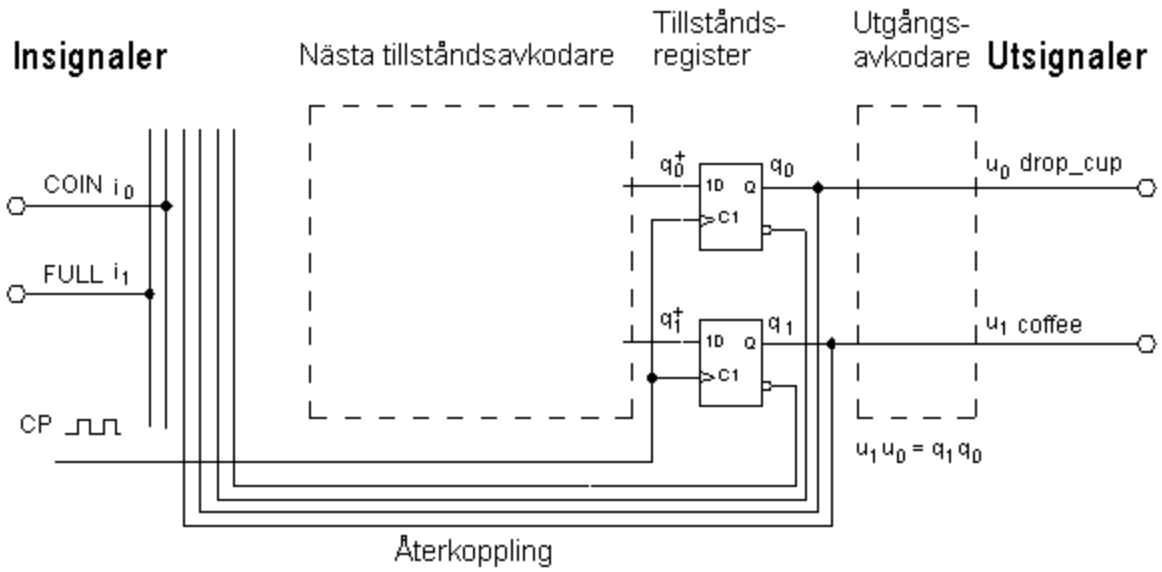


Lämplig klockfrekvens 1 Hz

I andra tillståndet, **Z1**, släpps plastmuggen (**`drop_cup`**). "1" betyder här "alltid uppfyllt" så nästa klockpuls (om 1 sek) lämnar man således **Z1** för **Z2**.

I det tredje tillståndet, **Z2**, återställs mataren för plastmuggar (**`!drop_cup`**) och koppen börjar fyllas med dryck (**`coffee`**). Detta pågår tills muggen är full (**`full`**). Vi hamnar då åter i utgångsläget.

Kaffeautomaten som Moore-automat



Tillståndsdiagrammet har tre tillstånd. **Tillståndsregistret** måste innehålla minst tre olika kombinationer. Det behövs två D-vippor ($2^2 = 4 > 3$).

Beteckningar. D-vippornas utgångar betecknas med $q_1\ q_0$, ingångarna, som bär nästa tillstånds signalen, brukar betecknas $q_1^+\ q_0^+$ där plustecknet står för det "kommande" tillståndet.
Sekvensnätets utgångar och ingångar betecknas $u_1\ u_0$, $i_1\ i_0$.

Om vippornas utgångar styr utsignalerna direkt slipper man utgångsavkodaren!

William Sandqvist william@kth.se

Kodad tillståndstabell

$$(q_1^+ q_0^+) = f(q_1 q_0, i_1 i_0)$$

$q_1 q_0$	I: $i_1 i_0$ 00 (!full !coin)	01 (!full coin)	11 (full coin)	10 (full !coin)
Z0: 00	0 0	0 1	0 0	0 0
Z1: 01	1 0	1 0	0 1	0 1
??: 11	0 0	0 0	0 0	0 0
Z2: 10	1 0	1 0	0 0	0 0

$q_1^+ q_0^+$

U=Z

$u_1 u_0 = q_1 q_0$

Z0: u_1 =!coffee, u_0 =!drop_cup

Z1: u_1 =!coffee, u_0 =drop_cup

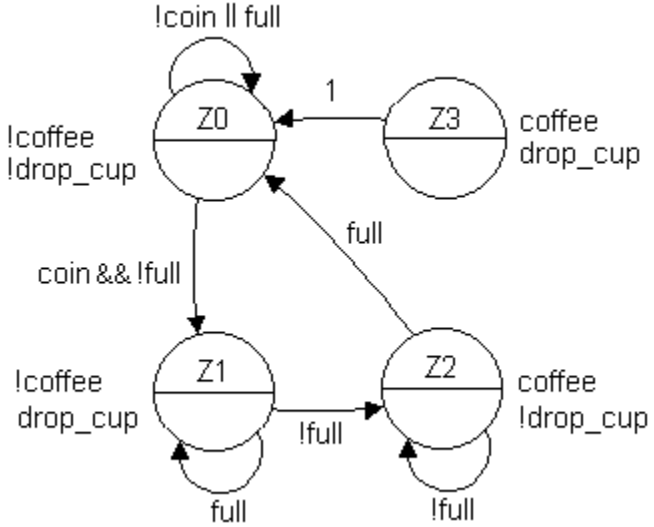
??: u_1 =coffee, u_0 =drop_cup

Z2: u_1 =coffee, u_0 =!drop_cup

Observera att tabellen är uppställd som ett Karnaughdiagram. Detta kommer att underlätta framtagandet av grindnäten senare!

Z2: I de två sista kolumnerna har koppen blivit full och vi ska gå till Z0 (00). I de två första kolumnerna stannar man kvar i Z2 (10) och fortsätter att fylla på dryck.

Ett mer genomtänkt tillståndsdiagram!

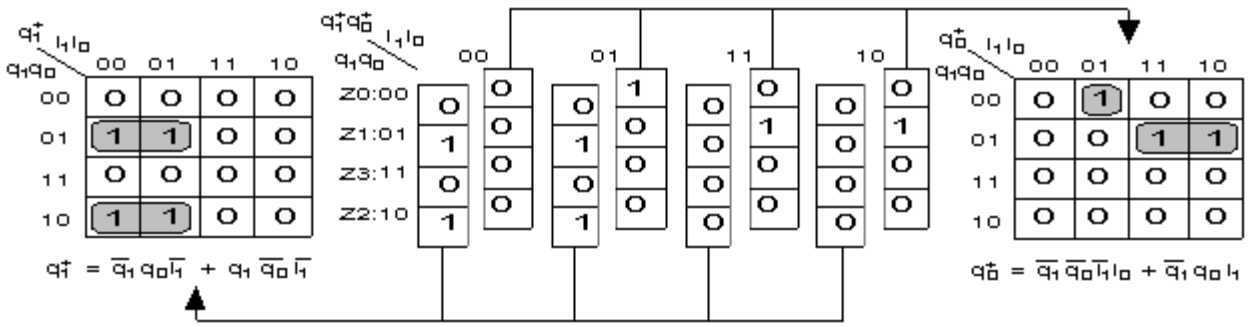


Detta enkla exempel visar den stora fördelen med en systematisk syntes-metod; man tvingas överväga ***alla*** möjligheter för att därmed se till att sekvensnätet gör det bästa av varje situation, även sådana som "egentligen" inte ska kunna hända (men som vi alla vet ***ändå*** kommer att hända …).

Vi kan nu "rätta till" det ursprungliga tillståndsdiagrammet.

Minimerade funktioner

Tabellen kan splittras upp i *två* Karnaughdiagram,
ett för $q_1^+ = f(\, q_1\, q_0\, ,\, i_1\, i_0\,)$ och ett för $q_0^+ = f(\, q_1\, q_0\, ,\, i_1\, i_0\,)$.

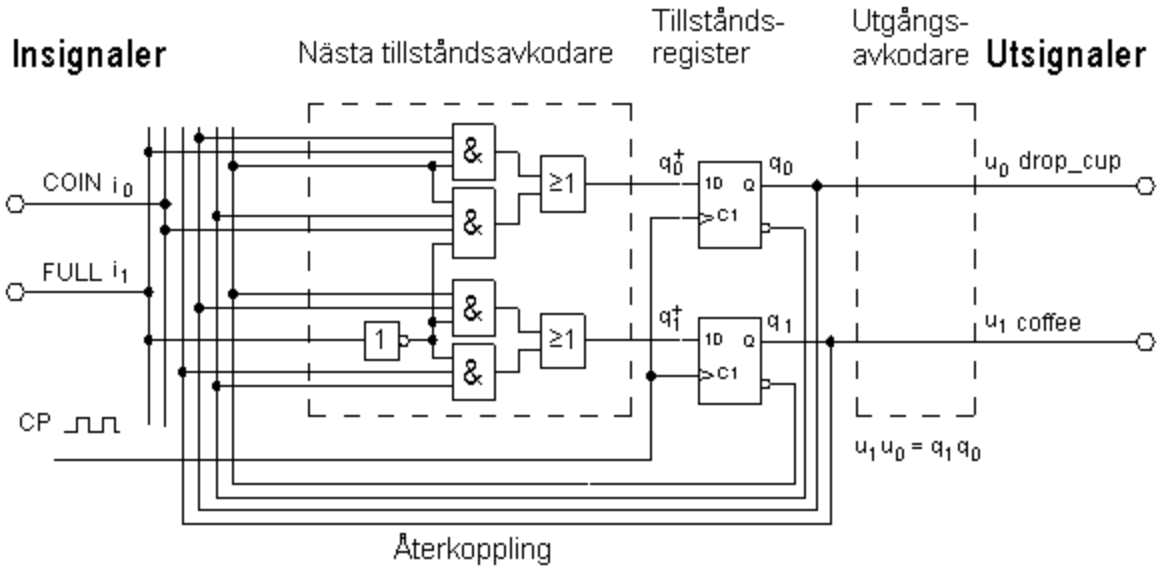


Hoptagningar i Karnaughdiagrammen ger:

$$q_1^+ = \bar{q}_1 q_0 \bar{i}_1 + q_1 \bar{q}_0 \bar{i}_1 \qquad q_0^+ = \bar{q}_1 \bar{q}_0 \bar{i}_1 i_0 + \bar{q}_1 q_0 i_1$$

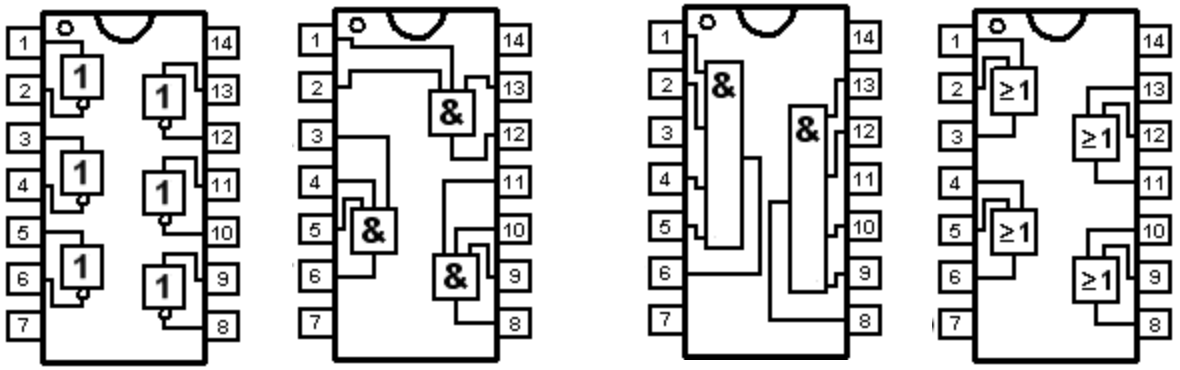
William Sandqvist william@kth.se

Kaffeautomat med standard-kretsar?



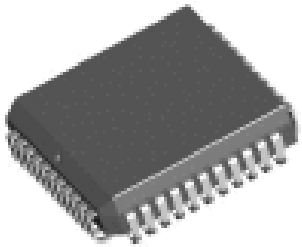
4 standardkretsar:

7404, 7411,
7421, 7432



William Sandqvist william@kth.se

Programmerbar logik VHDL?



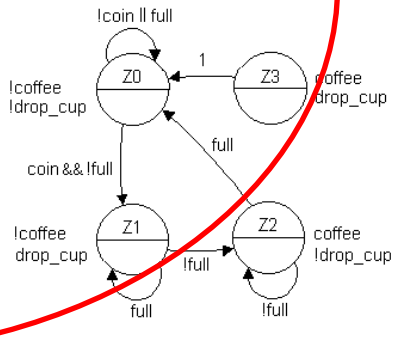
```
library IEEE;
use IEEE.std_logic_1164.all;

entity coffee_machine is
  port ( COIN, FULL       : in std_logic    ;
         DROP_CUP, COFFEE : out std_logic ) ;
end coffee_machine ;

architecture behavior of coffee_machine is
  type state is ( Z0, Z1, Z2, Z3 )          ;
  signal present_state, next_state : state ;

state_register: process ( CLK )
  begin
    if rising_edge( CLK ) then
        present_state <= next_state ;
    end if ;
  end process ;

output_decode: process ( present_state )
  begin
    case present_state is
      when Z0 => DROP_CUP <= '0' ;
                 COFFEE   <= '0' ;
      when Z1 => DROP_CUP <= '0' ;
                 COFFEE   <= '1' ;
      when Z2 => DROP_CUP <= '1' ;
                 COFFEE   <= '0' ;
      when Z3 => DROP_CUP <= '1' ;
                 COFFEE   <= '1' ;
    end case ;
  end process ;

next_state_decode: process ( present_state, COIN, FULL )
  begin
    case present_state is
      when Z0 => if (NOT COIN OR NOT FULL)    = '1' then
                   next_state <= Z0 ;
                 elsif (COIN AND NOT FULL) = '1' then
                   next_state <= Z1 ;
                 end if ;
      when Z1    => if FULL = '1' then
                   next_state <= Z1 ;
                 else
                   next_state <= Z2 ;
                 end if ;
      when Z2    => if FULL = '0' then
                   next_state <= Z2 ;
                 else
                   next_state <= Z0 ;
                 end if ;
      when Z3    =>   next_state <= Z3 ;
    end case ;
  end process ;
end architecture behavior ;
```

Entity

State register

Output decoder

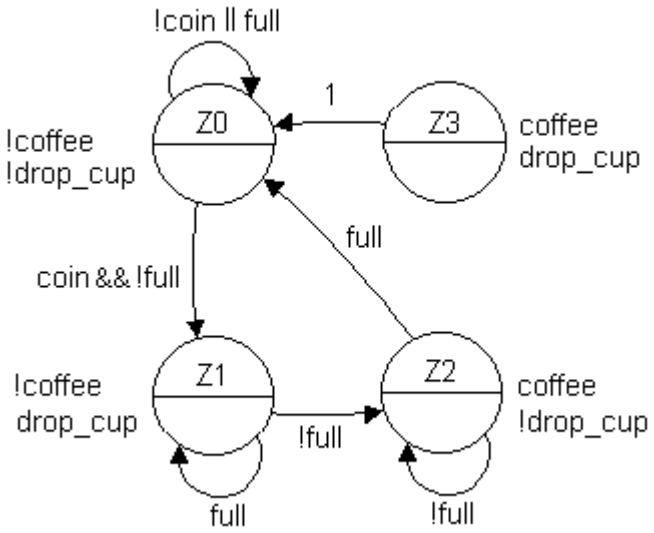
Next State Decoder

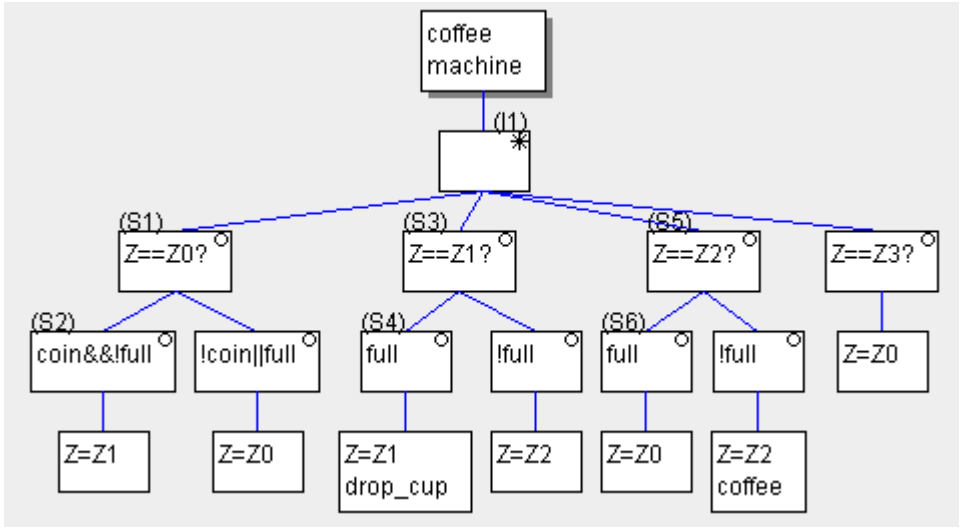


Kaffeautomaten direkt från tillståndsdiagrammets beskrivning.

William Sandqvist william@kth.se

Processor, C-kod från tillståndsdiagram



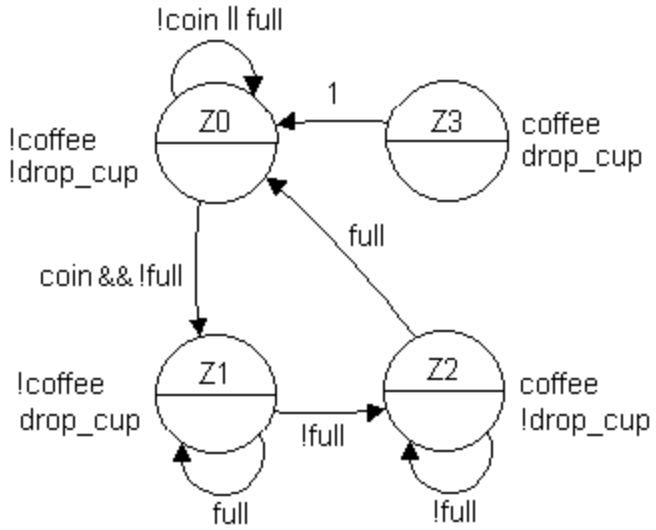


vanligen ”**`switch() case:`**” satser

William Sandqvist william@kth.se

Processor, C-kod från tillståndsdiagram

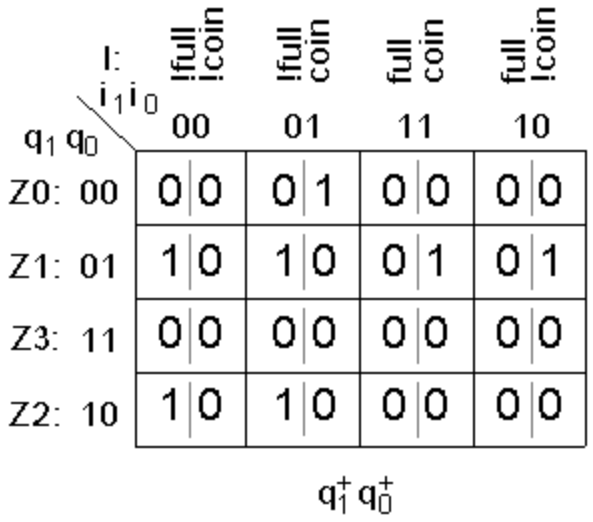
Vanligen använder man ”**`switch() case:`**” satser.




```
#define Z0 0
#define Z1 1
#define Z2 2
#define Z3 3
```

```
switch(state)
 {
   case Z0:
     if( coin||full  ) state = Z0;
     if( coin&&!full ) state = Z1;
     break;
   case Z1:
     if( full  ) state = Z1;
     if( !full ) state = Z2;
     break;
   case Z2:
     if( full  ) state = Z0;
     if( !full ) state = Z2;
     break;
   case Z3:
     state = Z0;
 }
```

Eller från tillståndstabell




```
index = state*4 + full*2 + coin;
```

index	$q_1 q_0 i_1 i_0$	$q_1^+ q_0^0$	Z	*index*	$q_1 q_0 i_1 i_0$	$q_1^+ q_0^0$	Z
0	0000	00	0	8	1000	10	2
1	0001	01	1	9	1001	10	2
2	0010	00	0	10	1010	00	0
3	0011	00	0	11	1011	00	0
4	0100	10	2	12	1100	00	0
5	0101	10	2	13	1101	00	0
6	0110	01	1	14	1110	00	0
7	0111	01	1	15	1111	00	0

```
int NextState[16]={0,1,0,0,2,2,1,1,2,2,0,0,0,0,0,0};
```

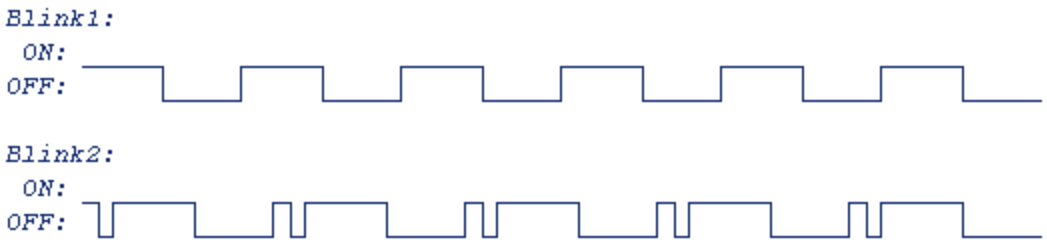
William Sandqvist william@kth.se

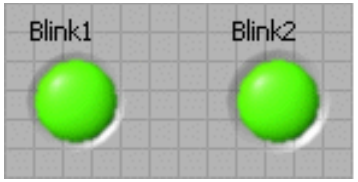
C-kod från tillståndstabell

Programmet blir mycket enkelt!

```
void main(void)
{
  int NextState[16]={0,1,0,0,2,2,1,1,2,2,0,0,0,0,0,0};
  . . . /* initiation */
  while(1)
    {
      full = GetFull();        /* sensor input */
      coin = GetCoin();        /* sensor input */
      index = state*4 + full*2 + coin;
      state = NextState[index];
      DropCup( state );    /* actuator output */
      FillCoffee( state ); /* actuator output */

      delay(1000);     /* 1 sec -> 1 Hz clock */
    }
}
```

Flera saker samtidigt?






```
/* Blink1:  1s ON - 1s OFF  */
/* Blink2:  0,2s ON - 0,2s OFF - 1s ON - 1s OFF  */
```

William Sandqvist william@kth.se

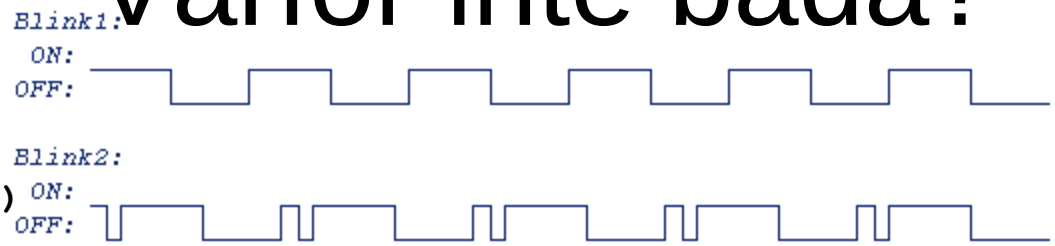
Först en lysdiod …


```
while(1)
 {
   /* Blink1:  1s ON - 1s OFF  */
   switch(State1)
    {
      case 0:
        Blink1 = ON;
        Time1++;
        if( Time1 == 10 ) { State1 = 1; Time1 = 0; }
        break;
      case 1:
        Blink1 = OFF;
        Time1++;
        if( Time1 == 10 ) { State1 = 0; Time1 = 0; }
    }
   delay(100); /* 0,1 sek delay */
 }
```

Sedan en annan lysdiod …


```
while(1)
 {
   /* Blink2:  0,2s ON - 0,2s OFF - 1s ON - 1s OFF */
   switch(State2)
    {
      case 0:
        Blink2 = ON; Time2++;
        if( Time2 == 2 ) { State2 = 1; Time2 = 0; }
        break;
      case 1:
        Blink2 = OFF; Time2++;
        if( Time2 == 2 ) { State2 = 2; Time2 = 0; }
        break;
      case 2:
        Blink2 = ON; Time2++;
        if( Time2 == 10 ) { State2 = 3; Time2 = 0; }
        break;
      case 3:
        Blink2 = OFF; Time2++;
        if( Time2 == 10 ) { State2 = 0; Time2 = 0; }
    }
   delay(100); /* 0,1 sek delay */
 }
```

Varför inte båda?




```
while(1)
 {
   /* Blink1:  1s ON - 1s OFF  */
   switch(State1)
    {
      case 0: ... ; break;
      case 1: ... ;
    }

   /* Blink2:  0,2s ON - 0,2s OFF - 1s ON - 1s OFF */
   switch(State2)
    {
      case 0:  ... ; break;
      case 1:  ... ; break;
      case 2:  ... ; break;
      case 3:  ... ;
    }

   delay(100); /* 0,1 sek delay */
 }
```

William Sandqvist william@kth.se

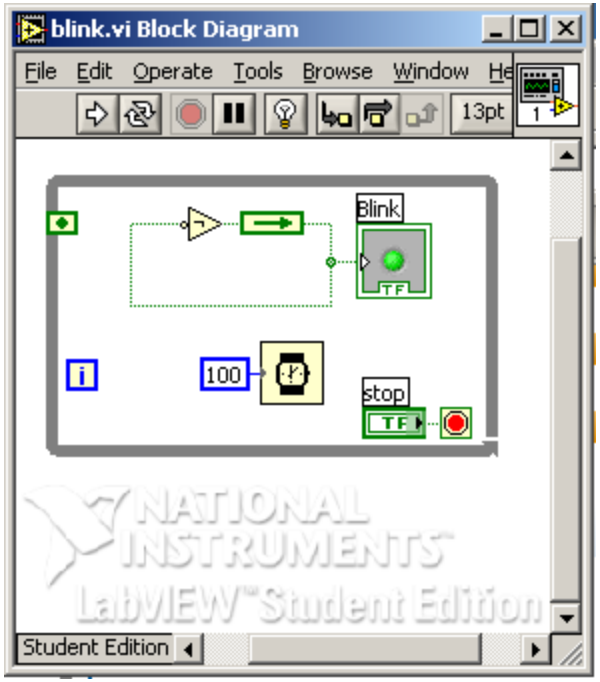
State machines

State machines/Tillståndsmaskiner/Moore-automater är således mycket användbara hjälpmedel för att programmera ”parallella” processer för små microcontrollers.

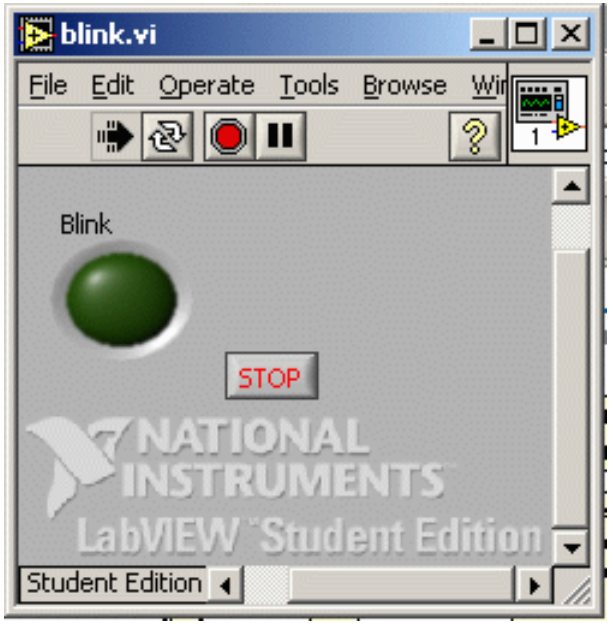
Tips! Observera att de flesta kompilatorer lägger ut effektivare kod för **`switch()`** – **`case`**
än för
`if()` – **`else if()`** – **`else`**
så använd alltid switch-satsen!

Många kommersiella programvaror bygger på State machines …

Grafiska programspråk, **LabView**




```
do {
     Blink != Blink;
     delay(1000); /* 1 sek delay */
   } while(!Stop );
```

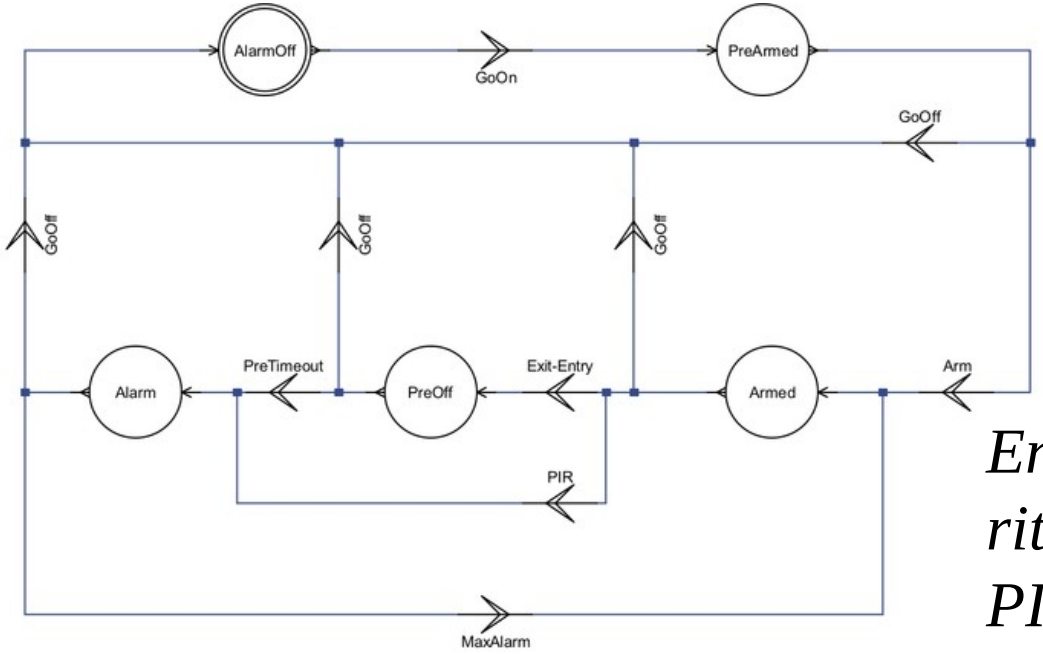


Bakom varje grafiskt element döljer sig en statemaskin …

William Sandqvist william@kth.se

Realizer®

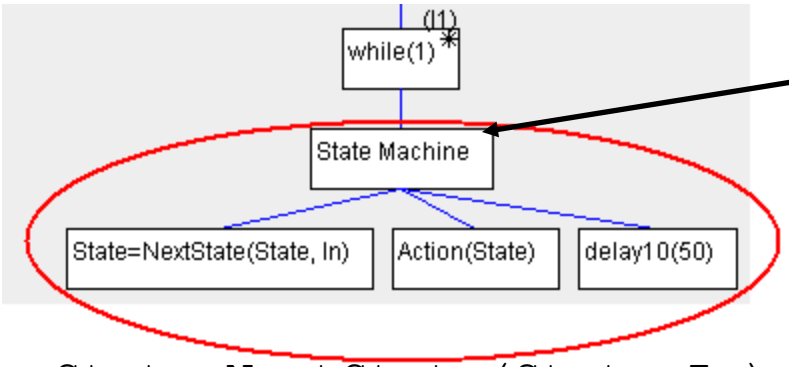
Realizer, ett grafiskt programmeringsverktyg för små processorer, tex. PIC.



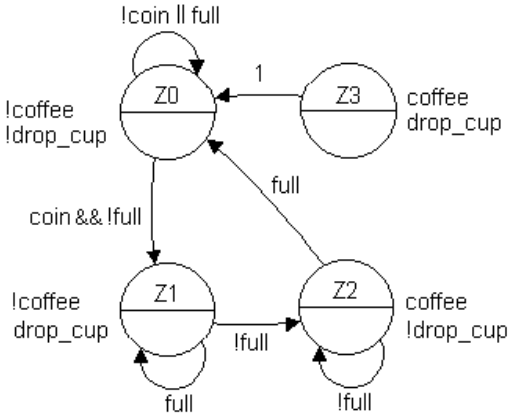
En statemachine ritad med Realizer. PIC-kod genereras sedan direkt.

William Sandqvist william@kth.se

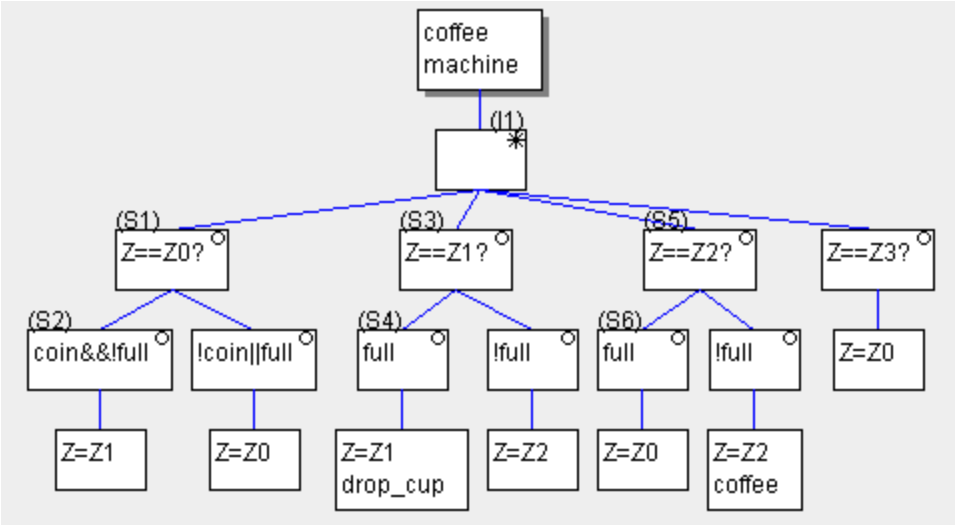
Dokumentation



State Machine




```
State=NextState(State,In)
Action(State)
```

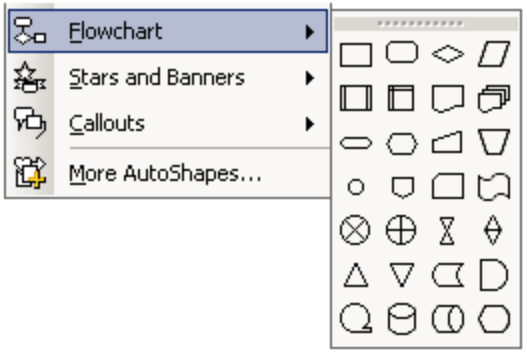


Det brukar bli tydligare om man kompletterar strukturdiagrammet med ett tillståndsdiagram, än om man försöker beskriva tillståndsövergångarna direkt i strukturdiagrammet.

William Sandqvist william@kth.se

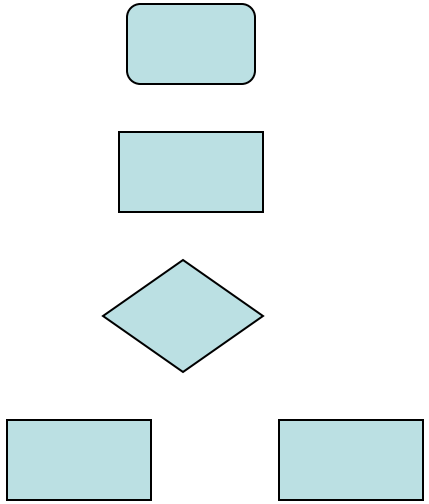
Flödesdiagram

Flödesdiagram använder man när det är viktigt att visa precis hur processorn exekverar koden. Till exempel när man analyserar assemblerkoden för att kunna se till att olika vägar genom programmet tar samma tid.



Verktyg för att rita flödesschemor finns i

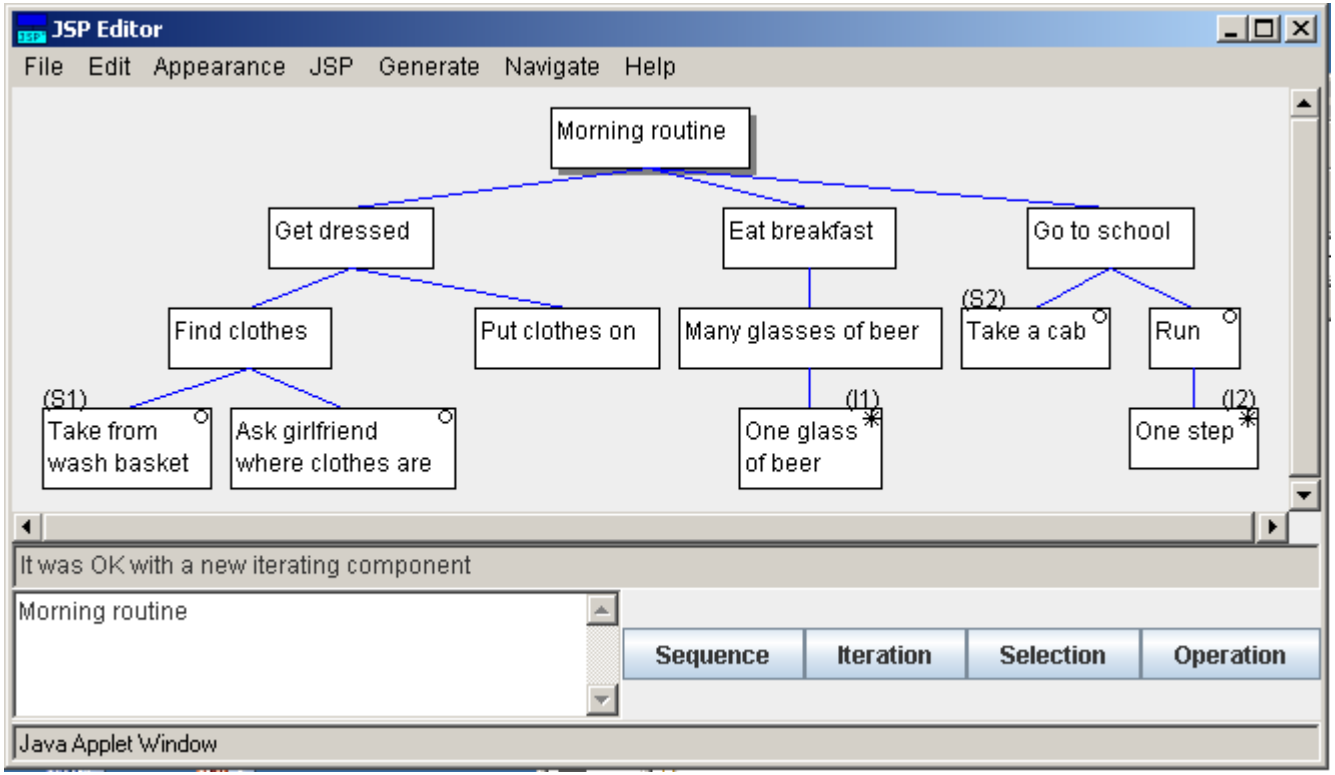
Word och **Powerpoint**.



Normalt använder man strukturdiagram för beskriva programstrukturer.

William Sandqvist william@kth.se

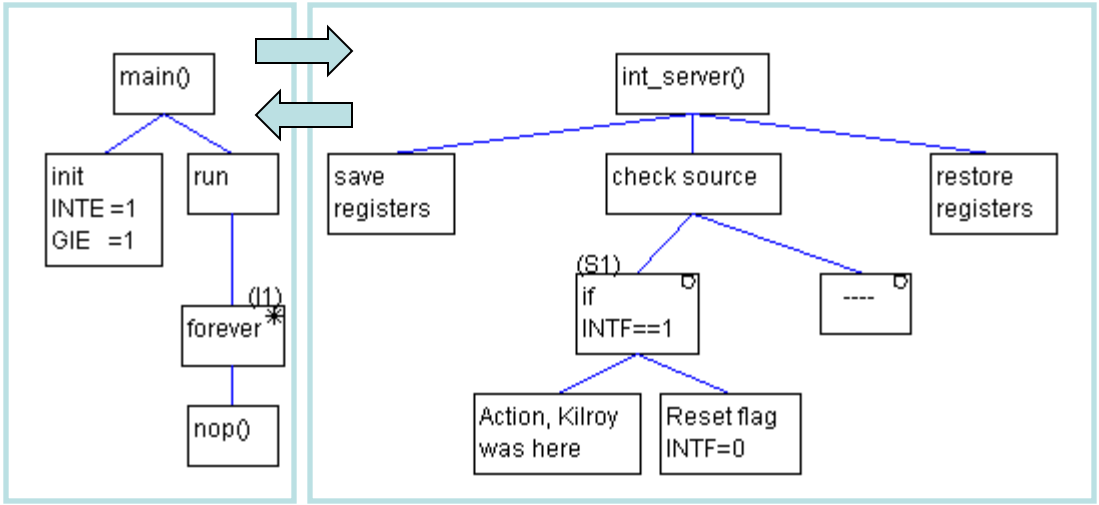
Strukturdiagram



Se ID120V: jsp.ppt JSP-editor som Java applet!

William Sandqvist william@kth.se

Interrupt?

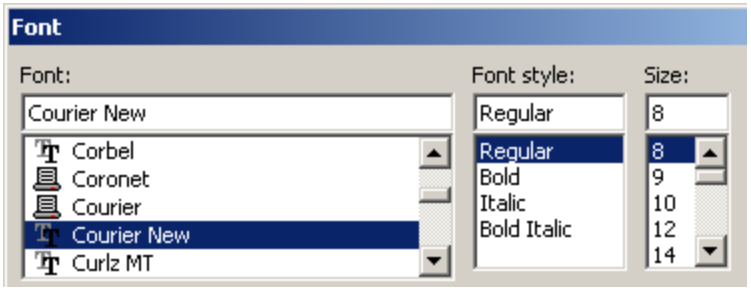


Interruptrutinen kan få ett fristående strukturdiagram.

William Sandqvist william@kth.se

Programkod

Programkod skrivs med ett typsnitt som har fast bredd (”skrivmaskinstil”). Vanliga typsnitt har annars variabel bredd: ”i” är ju smalare än ”W”.



Courier New, är ett sådant ”fast” typsnitt. Har Du använt 80 tecken per rad i texteditorn ska Size väljas till 8 punkter för att raderna ska få plats på A4. Om raderna bryts blir koden svårläst.

Tab-tecken kan behöva bytas ut mot mellanslag.

William Sandqvist william@kth.se

ASCII-grafik

```
/*
          ________ ________
         |        \/        |
         |RA2   16F628   RA1|->- PWMGND 1/0
         |RA3             RA0|
         |RA4         RA7/OSC1|
         |RA5/MCLR    RA6/OSC2|
 GND ---|Vss             Vdd|-- +5V
         |RB0/INT    (RB7)/PGD|
         |RB1/Rx     (RB6)/PGC|
2688 Hz  |RB2/Tx          RB5|
 PWM -<-|RB3/CCP     (RB4)/PGM|
         |_____________________|
Use LP-filter to se sinus
*/
```

Viktig information kan bakas in i programkoden som ASCII-grafik.

Den finns då tillgänglig för alla oavsett system.

William Sandqvist william@kth.se